Bimorphisms and synchronous grammars

Bimorphisms and synchronous grammars

Stuart M. Shieber
School of Engineering and Applied Sciences
Harvard University, Cambridge MA, USA

ABSTRACT

We tend to think of the study of language as proceeding by characterizing the strings and structures of a language, and we think of natural-language processing as using those structures to build systems of utility in manipulating the language. But many language-related problems are more fruitfully viewed as requiring the specification of a relation between two languages, rather than the specification of a single language.

In this paper, we provide a synthesis and extension of work that unifies two approaches to such language relations: the automata-theoretic approach based on tree transducers that transform trees to their counterparts in the relation, and the grammatical approach based on synchronous grammars that derive pairs of trees in the relation. In particular, we characterize synchronous tree-substitution grammars and synchronous tree-adjoining grammars in terms of bimorphisms, which have previously been used to characterize tree transducers. In the process, we provide new approaches to formalizing the various concepts: a metanotation for describing varieties of tree automata and transducers in equational terms; a rigorous formalization of tree-adjoining and tree-substitution grammars and their synchronous counterparts, using trees over ranked alphabets; and generalizations of tree-adjoining grammar allowing multiple adjunction.

Keywords:
synchronous grammars,
tree transducers,
tree-adjoining grammars,
tree-substitution grammars

1 INTRODUCTION

We tend to think of the study of language as proceeding by characterizing the strings and structures of a language, and we think of natural-language processing as using those structures to build systems of utility in manipulating the language. But many language-related problems are more fruitfully viewed as requiring the specification of a *relation* between two languages, rather than the specification of a single language. The paradigmatic case is machine translation, where the translation relation between the source and target natural languages is itself the goal to be characterized. Similarly, the study of semantics involves a relation between a natural language and a language of semantic representation (phonological form and logical form in one parlance). Computational interpretation of text, as in question-answering or natural-language command and control systems, requires computing that relation in the direction from natural language to semantic representation, and tactical generation in the opposite direction. Sentence paraphrase and compression can be thought of as computing a relation between strings of a single natural language. Similar examples abound.

The modelling of these relations has been a repeated area of study throughout the history of computational linguistics, proceeding in phases that have alternated between emphasizing automata-theoretic tools and grammatical tools. On the automata-theoretic side, the early pioneering work of Rounds (1970) on *tree transducers* was intended to formalize aspects of transformational grammars, and led to a long development of the formal-language theory of tree transducers. Grammatical approaches are based on the idea of synchronizing the grammars of the related languages. We use the general term *synchronous grammars* for the idea (Shieber and Schabes 1990), though early work in formalizing programming-language compilation uses the more domain-specific term *syntax-directed transduction* or *translation* (Lewis and Stearns 1968; Aho and Ullman 1969), and a variety of specific systems – inversion transduction grammars (Wu 1996, 1997), head transducers (Alshawi *et al.* 2000), multitext grammars (Melamed 2003, 2004) – forgo the use of the term. The early work on the synchronous grammar approach for natural-language application involved synchronizing tree-adjoining grammars (TAG). A recent

resurgence of interest in automata-theoretic approaches in the machine translation community (Graehl and Knight 2004; Galley *et al.* 2004) has led to more powerful types of transducers (Maletti *et al.* 2009) and a far better understanding of the computational properties of and relationships among different transducer types (Maletti *et al.* 2009). Synchronous grammars have also seen a rise in application in areas such as machine translation (Nesson *et al.* 2006; DeNeefe and Knight 2009), linguistic semantics (Nesson and Shieber 2006; Han and Hedberg 2008), and sentence compression (Yamangil and Shieber 2010).

As these various models were developed, the exact relationship among them had been unclear, with a large number of seemingly unrelated formalisms being independently proposed or characterized. In particular, the grammatical approach to tree relations found in synchronous grammar formalisms and the automata-theoretic approach of tree transducers have been viewed as contrasting approaches.

A reconciliation of these two approaches was initiated in two pieces of earlier work (Shieber 2004, 2006), which the present paper unifies, simplifies, and extends. That work proposed to use the formal-language-theoretic device of bimorphisms (Arnold and Dauchet 1982), previously little known outside the formal-language-theory community, as a means for unifying the two approaches and clarifying the interrelations. It investigated the formal properties of synchronous tree-substitution grammars (STSG) and synchronous tree-adjoining grammars (STAG) from this perspective, showing that both formalisms, along with traditional tree transducers, can be thought of as varieties of bimorphisms. This earlier work has already been the basis for further extensions, such as the synchronous context-free tree grammars of Nederhof and Vogler (2012).

The present paper includes all of the results of the prior two papers, with notations made consistent, presentations clarified and expanded, and proofs simplified, and therefore supersedes those papers. It provides a definitive presentation of the formal foundations for TSG, TAG, and their synchronous versions, improving on the earlier presentations. To our knowledge, it provides the most consistent definition of TAG and STAG available, and the only one to use trees over ranked rather than unranked alphabets. It also, in passing, provides a characterization of transducers in terms of equational systems using

a uniform metagrammar notation, a new characterization of the relation between tree-adjoining grammar derivation and derived trees, and a new simpler and more direct proof of the equivalence of tree-adjoining languages and the output languages of monadic macro tree transducers, formal contributions that may have independent utility. Finally, it extends the prior results to cover more linguistically appropriate variants of synchronous tree-adjoining grammars, in particular incorporating multiple adjunction.

After some preliminaries (Section 2), we present a set of known results relating context-free languages, tree homomorphisms, tree automata, and tree transducers to extend them for the tree-adjoining languages (Section 3), presenting these in terms of restricted kinds of functional programs over trees, using a simple grammatical notation for describing the programs. We review the definition of tree-substitution and tree-adjoining grammars (Section 4) and synchronous versions thereof (Section 5). We prove the equivalence between STSG and a variety of bimorphism (Section 6).

The grammatical presentation of transducers as functional programs allows us to easily express generalizations of the notions: monadic macro tree homomorphisms, automata, and transducers, which bear (at least some of) the same interrelationships that their traditional simpler counterparts do (Section 7). Finally, we use this characterization to place the synchronous TAG formalism in the bimorphism framework (Section 7.3), further unifying tree transducers and other synchronous grammar formalisms. We show that these methods generalize to TAG allowing multiple adjunction as well (Section 8).[1]

The present work, being based on and synthesizing work from some ten years ago, is by no means the last word in the general area. Indeed, since publication of the earlier articles, the connections among synchronous grammars, transducers, and bimorphisms have been considerably further clarified. The relation between bimorphisms and tree transducers has benefitted from a notion of extended top-down tree transducers, which have been shown to be strongly equivalent to the $B(LC, LC)$ bimorphism class we discuss below (Maletti 2008). Koller

[1] Much of the content in Sections 2–7 of this paper is based on material in previous papers (Shieber 2004, 2006), and is used by permission.

and Kuhlmann (2011) provide an elegant generalization of monolingual and synchronous systems in terms of interpreted regular tree grammars (IRTG), in spirit quite close to the idea here of reconstructing synchronous grammars as bimorphism-like formal systems. Their IRTG can be used for CFG, TSG, TAG, and synchronous versions of various sorts. Of especial interest are the formalizations of Büchse *et al.* (2012, 2014), which modify the definitions of TAG to incorporate state information at substitution and adjunction sites. This modification eliminates much of the inelegance of the formalization here that accounts for our having to couch the various equivalences we show in terms of weak rather than strong generative capacity. The presentation below should be helpful in understanding the background to these works as well.

2 PRELIMINARIES

We start by defining the terminology and notations that we will use for strings, trees, and the like.

2.1 *Basics*

We will notate sequences with angle brackets, e.g., $\langle a, b, c\rangle$, or where no confusion results, simply as abc, with the empty string written ϵ.

We follow much of the formal-language-theory literature (and in particular, the tree transducer literature) in defining trees over *ranked* alphabets, in which the symbols decorating the nodes are associated with fixed arities. (By contrast, formal work in computational linguistics typically uses unranked trees.) Trees will thus have nodes labeled with elements of a RANKED ALPHABET, a set of symbols $\mathcal{F}$, each with a non-negative integer RANK or ARITY assigned to it, determining the number of children for nodes so labeled. To emphasize the arity of a symbol, we will write it as a parenthesized superscript, for instance $f^{(n)}$ for a symbol f of arity n. Analogously, we write $\mathcal{F}^{(n)}$ for the set of symbols in $\mathcal{F}$ with arity n. Symbols with arity zero ($\mathcal{F}^{(0)}$) are called NULLARY symbols or CONSTANTS. The set of nonconstants is written $\mathcal{F}^{(\geq 1)}$.

To express incomplete trees, trees with "holes" waiting to be filled, we will allow leaves to be labeled with variables, in addition to nullary symbols. The set of TREES OVER A RANKED ALPHABET $\mathcal{F}$

AND VARIABLES $\mathcal{X}$, notated $\mathcal{T}(\mathcal{F}, \mathcal{X})$, is the smallest set such that

Nullary symbols at leaves $f \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}^{(0)}$;

Variables at leaves $x \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $x \in \mathcal{X}$;

Internal nodes $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}^{(n)}$, $n \geq 1$, and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Where convenient, we will blur the distinction between the leaf and internal node notation for a nullary symbol f, allowing $f()$ as synonymous for the leaf node f.

We abbreviate $\mathcal{T}(\mathcal{F}, \emptyset)$, where the set of variables is empty, as $\mathcal{T}(\mathcal{F})$, the set of GROUND TREES over $\mathcal{F}$. We will also make use of the set of n numerically ordered variables $\mathcal{X}_n = \{x_1, \ldots, x_n\}$, and write x, y, z as synonyms for x_1, x_2, x_3, respectively.

Trees can also be viewed as mappings from TREE ADDRESSES, sequences of integers, to the labels of nodes at those addresses. The address ϵ is the address of the root, 1 the address of the first child, 12 the address of the second child of the first child, and so forth. We write $q \prec p$ to indicate that tree address q is a proper prefix of p, and $p - q$ for the sequence obtained from p by removing prefix q from the front. For instance, $1213 - 12 = 13$.

We will use the notation t/p to pick out the subtree of the node at address p in the tree t, that is, (using $\cdot$ for the insertion of an element on a sequence)

$$t/\epsilon = t$$
$$f(t_1, \ldots, t_n)/(i \cdot p) = t_i/p \qquad \text{for } 1 \leq i \leq n \quad .$$

The notation $t@p$ picks out the label of the node at address p in the tree t, that is, the root label of t/p.

Replacing the subtree of t at address p by a tree t', written $t[p \mapsto t']$ is defined as

$$t[\epsilon \mapsto t'] = t'$$
$$f(t_1, \ldots, t_n)[(i \cdot p) \mapsto t'] = f(t_1, \ldots, t_i[p \mapsto t'], \ldots, t_n) \qquad \text{for } 1 \leq i \leq n \quad .$$

The HEIGHT of a tree t, notated $height(t)$, is defined as follows:

$$height(x) = 0 \qquad \text{for } x \in \mathcal{X}$$
$$height(f(t_1, \ldots, t_n)) = 1 + \max_{i=1}^{n} height(t_i) \qquad \text{for } f \in \mathcal{F}^{(n)}$$

We can use trees with variables as CONTEXTS in which to place other trees. A tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ will be called a context, typically denoted with the symbol C. The notation $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ denotes the tree in $\mathcal{T}(\mathcal{F})$ obtained by substituting for each x_i the corresponding t_i.

More formally, for a context $C \in \mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ and a sequence of n trees $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, the SUBSTITUTION OF $t_1, \ldots, t_n$ INTO C, notated $C[t_1, \ldots, t_n]$, is defined inductively as follows:

$$(f(u_1, \ldots, u_m))[t_1, \ldots, t_n] = f(u_1[t_1, \ldots, t_n], \ldots, u_m[t_1, \ldots, t_n])$$
$$x_i[t_1, \ldots, t_n] = t_i$$

2.2 *A grammatical metanotation*

We will use a grammatical notation akin to BNF to specify, among other constructs, equations defining functional programs of various sorts. As an introduction to this notation, here is a grammar defining trees over a ranked alphabet and variables (essentially identically to the definition given above):

$$\begin{aligned} f^{(n)} &\in \mathcal{F}^{(n)} \\ x \in \mathcal{X} &::= x_1 \mid x_2 \mid \cdots \\ t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) &::= f^{(m)}(t_1, \ldots, t_m) \\ &\mid \quad x \end{aligned}$$

The notation allows definition of classes of expressions (e.g., $\mathcal{F}^{(n)}$) and specifies metavariables over them ($f^{(n)}$). These classes can be primitive ($\mathcal{F}^{(n)}$) or defined ($\mathcal{X}$), even inductively in terms of other classes or themselves ($\mathcal{T}(\mathcal{F}, \mathcal{X})$). We use the metavariables and subscripted variants on the right-hand side to represent an arbitrary element of the corresponding class. Thus, the elements $t_1, \ldots, t_m$ stand for arbitrary trees in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and x an arbitrary variable in $\mathcal{X}$. Because numerically subscripted versions of x appear explicitly and individually enumerated as instances of $\mathcal{X}$ (on the right hand side of the rule defining variables), numerically subscripted variables (e.g., x_1) on the right-

hand side of all rules are taken to refer to the specific elements of $\mathcal{X}$ (for instance, in the definition (1) of tree transducers), whereas otherwise subscripted elements within the metanotation (e.g., x_i, t_1, t_m) are taken as metavariables.

3 TREE TRANSDUCERS, HOMOMORPHISMS, AND AUTOMATA

We review the formal definitions of tree transducers and related constructions for defining tree languages and relations, making use of the grammatical metanotation to define them as functional program classes.

3.1 *Tree transducers*

The variation in tree transducer formalisms is extraordinarily wide and the literature vast. For present purposes, we restrict attention to simple nondeterministic tree transducers operating top-down, which transform trees by replacing each node with a subtree as specified by the label of the node and the state of the transduction at that node.

Informally, a TREE TRANSDUCER (specifically a NONDETERMINISTIC TOP-DOWN TREE TRANSDUCER ($\downarrow T\,T$)) specifies a nondeterministic computation from $\mathcal{T}(\mathcal{F})$ to $\mathcal{T}(\mathcal{G})$ defined such that the symbol at the root of the input tree and a current state determines an output context in which the recursive images of the subtrees are placed. Formally, we can define a transducer as a kind of functional program, that is, a set of equations characterized by the following grammar for equations Eqn. (The set of states is conventionally notated Q, with members notated q. One of the states is distinguished as the INITIAL STATE of the transducer.)

$$
\begin{aligned}
q &\in Q \\
f^{(n)} &\in \mathcal{F}^{(n)} \\
g^{(n)} &\in \mathcal{G}^{(n)} \\
x \in \mathcal{X} &::= x_1 \mid x_2 \mid \cdots \\
Eqn &::= q(f^{(n)}(x_1, \ldots, x_n)) \doteq \tau^{(n)} \\
\tau^{(n)} \in \mathcal{R}^{(n)} &::= g^{(m)}({\tau^{(n)}}_1, \ldots, {\tau^{(n)}}_m) \\
&\mid \quad q_j(x_i) \quad \text{where } 1 \leq i \leq n
\end{aligned}
\tag{1}
$$

Intuitively speaking, the expressions in $\mathcal{R}^{(n)}$ are right-hand-side terms using variables limited to the first n.

Given this formal description of the set of equations Eqn, a tree transducer is defined as a tuple $\langle Q, \mathcal{F}, \mathcal{G}, \Delta, q_0 \rangle$ where[2]

- Q is a finite set of STATES;
- $\mathcal{F}$ is a ranked alphabet of INPUT SYMBOLS;
- $\mathcal{G}$ is a ranked alphabet of OUTPUT SYMBOLS;
- $\Delta \subseteq Eqn$ is a finite set of EQUATIONS;
- $q_0 \in Q$ is a distinguished INITIAL STATE.

Conventional nomenclature refers to the equations as TRANSITIONS, by analogy with transitions in string automata. We use both terms interchangeably. To make clear the distinction between these equations and other equalities used throughout the paper, we use the special equality symbol $\doteq$ for these equations.

The equations define a derivation relation as follows. Given a tree transducer $\langle Q, \mathcal{F}, \mathcal{G}, \Delta, q_0 \rangle$ and two trees $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{G} \cup Q)$ and $t' \in \mathcal{T}(\mathcal{F} \cup \mathcal{G} \cup Q)$, tree t DERIVES t' IN ONE STEP, notated $t \doteq t'$ if and only if there is an equation $u \doteq u' \in \Delta$ with $u \in \mathcal{T}(\mathcal{F} \cup Q, \mathcal{X}_n)$ and $u' \in \mathcal{T}(\mathcal{G} \cup Q, \mathcal{X}_n)$, and a tree $C \in \mathcal{T}(\mathcal{F} \cup \mathcal{G} \cup Q, \mathcal{X}_1)$ in which the variable x_1 occurs exactly once and trees $u_1, \ldots, u_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{G})$, such that

$$t = C[u[u_1, \ldots, u_n]]$$

and

$$t' = C[u'[u_1, \ldots, u_n]] \qquad .$$

We abuse notation by using the same symbol for the transition equations and the one-step derivation relation they define, and will further extend the abuse to cover the derivation relation's reflexive transitive closure.

The TREE RELATION defined by a $\downarrow TT$ $\langle Q, \mathcal{F}, \mathcal{G}, \Delta, q_0 \rangle$ is the set of all tree pairs $\langle s, t \rangle \in \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{G})$ such that $q_0(s) \doteq t$. By virtue of nondeterminism in the equations, multiple equations for a given state q and symbol f, tree transducers define true relations rather than merely functions.

[2] We assume without loss of generality that $\mathcal{F}$, $\mathcal{G}$, and Q are disjoint so that their union can itself be taken to be a well-formed ranked alphabet. The elements of the set Q are taken to be ranked symbols of arity 1.

By way of example, the equation grammar above allows the definition of the following set of equations defining a tree transducer:[3]

$$q(f(x)) \doteq g(q'(x), q(x))$$
$$q(a) \doteq a$$

$$q'(f(x)) \doteq f(q'(x))$$
$$q'(a) \doteq a$$

This transducer allows for the following derivation:

$$\begin{aligned} q(f(f(a))) &\doteq g(q'(f(a)), q(f(a))) \\ &\doteq g(f(q'(a)), g(q'(a), q(a))) \\ &\doteq g(f(a), g(a, a)) \end{aligned}$$

3.2 *Subvarieties of transducers*

Important subvarieties of the basic transducers can be defined by restricting the trees τ that form the right-hand sides of equations, the elements of $\mathcal{R}^{(n)}$ used.

Recall that each equation is of the form

$$q(f^{(n)}(x_1, \ldots, x_n)) \doteq \tau^{(n)} \quad .$$

A transducer is

- LINEAR if for each such equation defining the transducer, τ is linear, that is, no variable is used more than once;
- COMPLETE if τ contains every variable in $\mathcal{X}_n$ at least once;
- ϵ-FREE if $\tau \notin \mathcal{X}_n$;
- SYMBOL-TO-SYMBOL if $height(\tau) = 1$; and
- a DELABELING if τ is complete, linear, and symbol-to-symbol.

[3] We will, in general, leave off the explicit specification of the set of states, input and output ranked alphabet, and initial state when providing example transducers, in the expectation that the sets of states and symbols can be inferred from the equations, and the initial state determined under a convention that it is the state defined in the textually first equation.

Note also that we avail ourselves of consistent renaming of the variables x_1, x_2, and so forth, where convenient for readability.

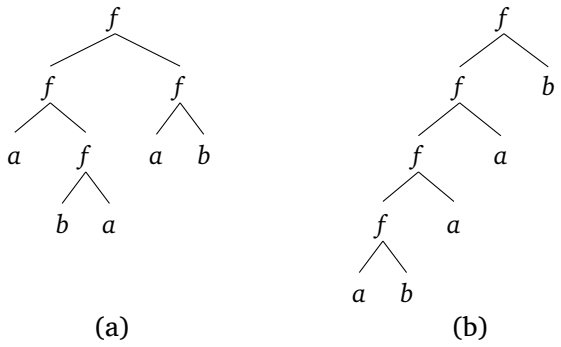


Figure 1:
Local rotation computed by a nonlinear tree transducer. Trees (a) and (b) are in the tree relation of the transducer defined in Section 3.3.

3.3 *Nonlinearity deprecated*

The following rules specify a transducer that recursively "rotates" subtrees of the form $f(t_1, f(t_2, t_3))$ to the tree $f(f(t_1, t_2), t_3)$, failing if the required pattern is not found.

$$\begin{aligned} q(f(x,y)) &\doteq f(f(q(x), q_1(y)), q_2(y)) \\ q_1(f(x,y)) &\doteq q(x) \\ q_2(f(x,y)) &\doteq q(y) \\ q(a) &\doteq a \\ q(b) &\doteq b \end{aligned}$$

The tree $f(f(a, f(b,a)), f(a,b))$ is transduced to $f(f(f(f(a,b),a),a),b)$ (as depicted graphically in Figure 1) according to the following derivation:

$$\begin{aligned} & q(f(f(a, f(b,a)), f(a,b))) \\ & \quad \doteq \quad f(f(q(f(a, f(b,a))), q_1(f(a,b))), q_2(f(a,b))) \\ & \quad \doteq \quad f(f(f(f(q(a), q_1(f(b,a))), q_2(f(b,a))), q(a)), q(b)) \\ & \quad \doteq \quad f(f(f(f(a, q(b)), q(a)), a), b) \\ & \quad \doteq \quad f(f(f(f(a,b),a),a),b) \end{aligned}$$

A variant transducer can allow f subtrees to remain unchanged (rather than failing) when the second argument is not itself an f tree. We add a (nondeterministic) equation to allow nonrotation,

$$q(f(x,y)) \doteq f(q(x), q'(y)) \qquad ,$$

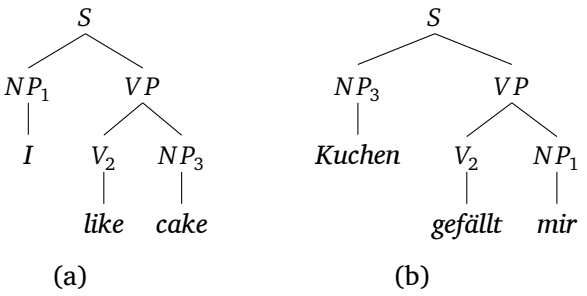


Figure 2: Example of local rotation in language translation divergence. Corresponding nodes are marked with matched subscripts.

which puts the proper constraint on its second subtree y through the new state q' defined by

$$q'(a) \doteq a$$
$$q'(b) \doteq b \quad .$$

This allows, for instance, the "already rotated" tree in Figure 1(b) to transduce to itself.

Note that intrinsic use is made in these examples of the ability to duplicate variables on the right-hand sides of rewrite rules. Transducers without such duplication are *linear*. Linear tree transducers are incapable of performing local rotations of this sort.

Local rotations are typical of natural-language applications. For instance, many of the kinds of translation divergences between languages, such as that exemplified in Figure 2, manifest such rotations. Similarly, semantic bracketing paradoxes can be viewed as necessitating rotations. Thus, linear tree transducers are insufficient for natural-language modeling purposes.

Nonlinearity per se, the ability to make copies during transduction, is not the kind of operation that is characteristic of natural-language phenomena. Furthermore, nonlinear transducers are computationally problematic. The following nonlinear transducer generates a tree that doubles in both width and depth.

$$q(f(x)) \doteq g(f(f(q(x))), f(f(q(x))))$$
$$q(g(x,y)) \doteq g(q(x), q(y))$$
$$q(a) \doteq a$$

For instance, the tree $f(a)$ transduces to

$$g(f(f(a)), f(f(a)))$$

which in turn transduces to

$$\begin{aligned} &g(g(f(f(g(f(f(a)),f(f(a))))),\\ &\quad f(f(g(f(f(a)),f(f(a))))),\\ &\quad g(f(f(g(f(f(a)),f(f(a))))),\\ &\quad f(f(g(f(f(a)),f(f(a)))))) \end{aligned} \quad .$$

Notice that the number of a's in the i-th iteration is 2^{2^i-1}. The size of this transducer's output is exponential in the size of its input. (The existence of such a transducer constitutes a simple proof of the lack of composition closure of tree transducers, as the exponential of an exponential grows faster than exponential.)

In summary, nonlinearity seems inappropriate on computational and linguistic grounds, yet is required for tree transducers to express the kinds of simple local rotations that are typical of natural-language transductions. By contrast, STSG, as described in Section 6, is intrinsically a linear formalism but can express rotations straightforwardly.

3.4 *Tree automata and homomorphisms*

Two subcases of tree transducers are especially important. First, tree transducers that implement a partial identity function over their domain are TREE AUTOMATA. These are delabeling tree transducers that preserve the label and the order of arguments. Because they compute only the identity function, tree automata are of interest for the domains over which they are defined, not the mappings they compute. This domain forms a tree language, the tree language recognized by the automaton. The tree languages so recognized are the REGULAR TREE LANGUAGES (or RECOGNIZABLE TREE LANGUAGES). Though the regular tree languages are a superset of the tree languages defined by context-free grammars (the local tree languages), the string languages defined by their yield are coextensive with the context-free languages. We take tree automata to be quadruples by dropping one of the redundant alphabets from the corresponding tree transducer quintuple.

Second, TREE HOMOMORPHISMS are deterministic tree transducers with only a single state, hence essentially stateless. The replacement of a node by a subtree thus proceeds deterministically and independently of its context. Consequently, a homomorphism $h : \mathcal{T}(\mathcal{F}) \rightarrow$

$\mathcal{T}(\mathcal{G})$ is specified by its kernel, a function $\hat{h} : \mathcal{F} \rightarrow \mathcal{T}(\mathcal{G}, \mathcal{X}_\infty)$ such that $\hat{h}(f)$ is a context in $\mathcal{T}(\mathcal{G}, \mathcal{X}_{arity(f)})$ for each symbol $f \in \mathcal{F}$. The kernel $\hat{h}$ is extended to the homomorphism h by the following recurrence:

$$h(f(t_1, \ldots, t_n)) = \hat{h}(f)[h(t_1), \ldots, h(t_n)]$$

that is, $\hat{h}(f)$ acts as a context in which the homomorphic images of the subtrees are substituted.

As with transducers (see Section 3.2), further restrictions can be imposed to generate the subclasses of linear, complete, ϵ-free, symbol-to-symbol, and delabeling tree homomorphisms.

The import of these two subcases of tree transducers lies in the fact that the tree relations defined by certain tree transducers have been shown to be also characterizable by composition from these simplified forms, via an alternate and quite distinct formalization, to which we now turn.

3.5 *The bimorphism characterization of tree transducers*

Tree transducers can be characterized directly in terms of equations defining a simple kind of functional program, as above. Bimorphisms constitute an elegant alternative characterization of tree transducers in terms of a constellation of elements of the various subtypes of transducers – homomorphisms and automata – we have introduced.

A BIMORPHISM is a triple $\langle L, h_{in}, h_{out} \rangle$ consisting of a regular tree language L (or, equivalently, a tree automaton) and two tree homomorphisms h_{in} and h_{out} (connoting the input and output respectively). The tree relation $\mathcal{L}$ defined by a bimorphism is the set of tree pairs that are generable from elements of the tree language by the homomorphisms, that is,

$$\mathcal{L}(\langle L, h_{in}, h_{out} \rangle) = \{\langle h_{in}(t), h_{out}(t) \rangle \mid t \in L\} \quad .$$

Depending on the type of tree homomorphisms used in the bimorphism, different classes of tree relations are defined. We can limit attention to bimorphisms in which the input or output homomorphisms are restricted to a certain type: linear (L), complete (C), ϵ-free (F), symbol-to-symbol (S), delabeling (D), or unrestricted (M). We will write $B(I, O)$ where I and O characterize a subclass of homomorphisms for the set of bimorphisms for which the input homomorphism is in the

subclass indicated by I and the output homomorphism is in the subclass indicated by O. For example, $B(D, M)$ is the set of bimorphisms for which the input homomorphism is a delabeling but the output homomorphism can be arbitrary.

The tree relations definable by bottom-up tree transducers (closely related to the top-down transducers we use here) turn out to be exactly this class $B(D, M)$. (See the survey by Comon *et al.* (2008, Section 6.5) and works cited therein.) The bimorphism notion thus allows us to characterize certain tree transductions purely in terms of tree automata and tree homomorphisms.

As an example, we consider the rotation transducer of Section 3.3, expressed as a bimorphism. The tree relation for the bimorphism expresses an abstract specification of where the rotations are to occur, picking out such cases with a special symbol R of arity 3, its arguments being the three subtrees participating in the rotation.

$$\begin{aligned} q(A) &\doteq A \\ q(B) &\doteq B \\ q(R(x, y, z)) &\doteq R(q(x), q(y), q(z)) \end{aligned}$$

The input homomorphism maps these trees onto trees prior to rotation.

$$\begin{aligned} q(A) &\doteq a \\ q(B) &\doteq b \\ q(R(x, y, z)) &\doteq f(q(x), f(q(y), q(z))) \end{aligned}$$

Notice that the trees rooted in R map onto a tree configuration that should be rotated.

The output homomorphism maps each tree onto the corresponding post-rotation tree:

$$\begin{aligned} q(A) &\doteq a \\ q(B) &\doteq b \\ q(R(x, y, z)) &\doteq f(f(q(x), q(y)), q(z)) \end{aligned}$$

Again, to allow the option of nonrotating configurations, we can add to the control trees nodes labeled F that should map onto configurations that cannot be rotated. (New equations are marked with $\Leftarrow$.)

The new q' state guarantees this constraint on the F trees.

$$\begin{aligned}
q(A) &\doteq A \\
q(B) &\doteq B \\
q(F(x,y)) &\doteq F(q(x), q'(y)) && \Leftarrow \\
q(R(x,y,z)) &\doteq R(q(x), q(y), q(z)) \\
q'(A) &\doteq A && \Leftarrow \\
q'(B) &\doteq B && \Leftarrow
\end{aligned}$$

The input homomorphism maps the new F states onto f trees

$$\begin{aligned}
q(A) &\doteq a \\
q(B) &\doteq b \\
q(F(x,y)) &\doteq f(q(x), q(y)) && \Leftarrow \\
q(R(x,y,z)) &\doteq f(q(x), f(q(y), q(z)))
\end{aligned}$$

as does the output homomorphism.

$$\begin{aligned}
q(A) &\doteq a \\
q(B) &\doteq b \\
q(F(x,y)) &\doteq f(q(x), q(y)) && \Leftarrow \\
q(R(x,y,z)) &\doteq f(f(q(x), q(y)), q(z))
\end{aligned}$$

4 TREE-SUBSTITUTION AND TREE-ADJOINING GRAMMARS

Tree-adjoining grammars (TAG) and tree-substitution grammars (TSG) are grammar formalisms based on tree rewriting, rather than the string rewriting of the Chomsky hierarchy formalisms. Grammars are composed of a set of elementary trees, which are combined according to simple tree operations. In the case of TAG, these operations are substitution and adjunction, in the case of TSG, substitution alone. Synchronous variants of these formalisms extend the base formalism with the synchronization idea presented in earlier work (Shieber 1994). In particular, grammars are composed of pairs of elementary trees, and certain pairs of nodes, one from each tree in a pair, are linked to indi-

cate that operations incorporating trees from a single elementary pair must occur at the linked nodes.

We review here the definition of tree-substitution and tree-adjoining grammars, and their synchronous variants. Since TSG can be thought of as a subset of TAG, we first present TAG, describing the restriction to TSG thereafter. Our presentation of TAG differs slightly from traditional ones in ways that simplify the synchronous variants and the later bimorphism constructions.

4.1 *Tree-adjoining grammars*

A tree-adjoining grammar is composed of a set of elementary trees, such as those depicted in Figure 4, that are combined by operations of substitution and adjunction. Traditional presentations of TAG, with which we will assume familiarity, take the symbols in elementary and derived trees to be unranked; nodes labeled with a given nonterminal symbol may have differing numbers of children. (Joshi and Schabes (1997) present a good overview.) For example, foot nodes of auxiliary trees and substitution nodes have no children, whereas the similarly labeled root nodes must have at least one. Similarly, two nodes with the same label but differing numbers of children may match for the purpose of allowing an adjunction (as the root nodes of α_1 and β_1 in Figure 4). In order to integrate TAG with tree transducers, however, we move to a ranked alphabet, which presents some problems and opportunities. (In some ways, the ranked alphabet definition of TAGs is slightly more elegant than the traditional one.)

We will thus take the nodes of TAG trees to be labeled with symbols from a ranked alphabet $\mathcal{F}$; a given symbol then has a fixed arity and a fixed number of children. However, in order to maintain information about which symbols may match for the purpose of adjunction and substitution, we take the elements of $\mathcal{F}$ to be explicitly formed as pairs of an unranked label e and an arity n. (For notational consistency, we will use e for unranked and f for ranked symbols.) We will notate these elements, abusing notation, as $e^{(n)}$, and make use of a function $|\cdot|$ to unrank symbols in $\mathcal{F}$, so that $|e^{(n)}| = e$.

To handle foot nodes, for each non-nullary symbol $e^{(i)} \in \mathcal{F}^{(\geq 1)}$, we will associate a new nullary symbol e_*, which one can take to be the pair of e and $*$; the set of such symbols will be notated $\mathcal{F}_*$. Similarly, for substitution nodes, $\mathcal{F}_\downarrow$ will be the set of nullary symbols $e_\downarrow$

for all $e^{(i)} \in \mathcal{F}^{(\geq 1)}$. These additional symbols, since they are nullary, will necessarily appear only at the frontier of trees. We will extend the function $|\cdot|$ to provide the unranked symbol associated with these symbols as well, so $|e_{\downarrow}| = |e_{*}| = e$.

A TAG grammar (which we will define more precisely shortly) is based then on a set P of elementary trees, a finite subset of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\downarrow} \cup \mathcal{F}_{*})$, divided into the auxiliary and initial trees depending on whether they do or do not possess a foot node, respectively. In order to allow reference to a particular tree in the set P, we associate with each tree a unique name, conventionally notated with a subscripted α or β for initial and auxiliary trees respectively. We will abuse notation by using the name and the tree that it names interchangably, and use primed and subscripted variants of α and β as variables over initial and auxiliary trees, with γ serving for elementary trees in general.

Traditionally in TAG grammars, substitutions are obligatory at substitution nodes (those with labels from $\mathcal{F}_{\downarrow}$) and adjunctions are optional at nodes with labels from $\mathcal{F}$. This presents two problems. First, the optionality of adjunction makes it tricky to provide a canonical fixed-length specification of what trees operate at the various nodes in the tree; such a specification will turn out to be helpful in our definitions of derivation for TAG and synchronous TAG. (This is not a problem for substitution, as the obligatoriness of substitution means that there will be exactly as many trees substituting in as there are substitution nodes.) Second, it is standard within TAG to provide further constraints that disallow adjunction at certain nodes. So far, we have no provision for such NONADJOINING CONSTRAINTS. To address these problems, we use a TAG formalism slightly modified from traditional presentations, one that loses no expressivity in weak generative capacity but is easier for analysis purposes.

First, we make all adjunction obligatory, in the sense that if a node in a tree allows adjunction, an adjunction must occur there. To get the effect of optional adjunction, for instance at a node labeled B, we add to the grammar a NONADJUNCTION TREE NA_B, a vestigial auxiliary tree of a single node B_{*}, which has no adjunction sites and therefore does not itself modify any tree that it adjoins into. These nonadjunction trees thus found the recursive structure of derivations.[4]

[4] In traditional TAG, all adjunction is optional; adding nonadjunction trees

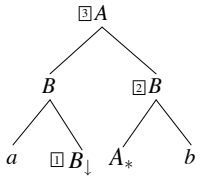


Figure 3:
Sample TAG tree marked with diacritics to show the permutation of operable nodes. Note that the node at address 1 is left out of the set of operable sites; it is thus a nonadjoining node.

Second, now that it is determinate whether an operation must occur at a node, the number of children of a node in a derivation tree is determined by the elementary tree γ at that node; it is just the number of adjunction or substitution sites in γ, the OPERABLE SITES, which we will notate $\overline{\gamma}$. We take $\overline{\gamma}$ to be the set of adjunction and substitution nodes in the tree, that is, all nodes in the tree with the exception of the foot node. (Below, we will allow for nodes to be left out from the set of operable sites, and in Section 8, we generalize this to allow multiple adjunctions at a single site.)

All that is left is to provide a determinate ordering of operable sites in an elementary tree, that is, a permutation π on the operable sites $\overline{\gamma}$ (or equivalently, their addresses). This permutation can be thought of as specified as part of the elementary tree itself. For example, the tree in Figure 3, which requires operations at the nodes at addresses ϵ, 12, and 2, may be associated with the permutation $\langle 12, 2, \epsilon \rangle$. The permutation can be marked on the tree itself with numeric diacritics $\boxed{i}$, as shown in the figure.

A nonadjoining constraint on a node can now be implemented merely by removing the node from the operable sites of a tree, and hence from the tree's associated permutation. In the graphical depictions, nonadjoining nodes are those non-substitution nodes that bear no numeric diacritic.

Formally, we define $\mathcal{E}(\mathcal{F})$, the ELEMENTARY TREES over a ranked alphabet $\mathcal{F}$, to be all pairs ${}_{\square}\gamma = \langle \gamma, \pi \rangle$ where $\gamma \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\downarrow} \cup \mathcal{F}_{*})$ and π is a permutation of a subset of the nodes in γ. As above, we use the notation $\overline{\gamma}$ to specify the operable sites of γ, that is, the domain of π. The operable sites $\overline{\gamma}$ must contain all substitution nodes in γ.

---

for all elements of $\mathcal{F}$ is consistent with that practice. Our approach, however, opens the possibility of leaving out nonadjunction trees for one or more symbols, thereby implementing a kind of global obligatory adjunction constraint, less expressive than those variants of TAG that have node-based obligatory adjunction constraints, but more so than the purely adjunction-optional approach.

We further require that the tree γ whose root is labeled f contain at most one node labeled with $|f|_* \in \mathcal{F}_*$ and no other nodes labeled in $\mathcal{F}_*$; this node is its foot node, and its address is notated $\text{foot}(\beta)$. The foot node is not an element of $\overline{\gamma}$. Trees with a foot node are AUXILIARY TREES; those with no foot node are INITIAL TREES. The set $\mathcal{E}(\mathcal{F})$ is the set of all possible such elementary trees.

The notation ${}_{\square}\gamma$ is used to indicate an elementary tree, the box as a mnemonic for the box diacritics labeling the permutation. We use similar notations for the particular cases where the elementary tree is initial (${}_{\square}\alpha$) or auxiliary (${}_{\square}\beta$). For convenience, for an elementary tree ${}_{\square}\gamma$, we will use γ for its tree component when no confusion results, and will conflate the tree properties of an elementary tree ${}_{\square}\gamma$ and its tree component γ.

A TAG grammar is then a triple $\langle \mathcal{F}, P, S \rangle$, where $\mathcal{F}$ is a ranked alphabet; P is the set of ELEMENTARY TREES, a finite subset of $\mathcal{E}(\mathcal{F})$; and $S \in \mathcal{F}_\downarrow$ is a distinguished INITIAL SYMBOL. We further partition the set P into the set I of initial trees in P and the set A of auxiliary trees in P. A simple TAG grammar is depicted in Figure 4; α_1 and α_2 are initial trees, and β_1 and β_2 are auxiliary trees.

4.2 *The substitution and adjunction operations*

We turn now to the operations used to derive more complex trees from the elementary trees. It is convenient to notationally distinguish derived trees that have the *form* of an initial or auxiliary tree, that is, (respectively) lacking or bearing a foot node. We use the bolded symbols $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ for derived trees in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_\downarrow \cup \mathcal{F}_*)$ without and with foot nodes, respectively, again using $\boldsymbol{\gamma}$ when being agnostic as to the form.

The trees are combined by two operations, SUBSTITUTION and ADJUNCTION. Under substitution, a node labeled $e_\downarrow$ (at address p) in a tree $\boldsymbol{\gamma}$ can be replaced by an initial-form tree $\boldsymbol{\alpha}$ with the corresponding label f at the root when $|f| = e$. The resulting tree, the substitution of $\boldsymbol{\alpha}$ at p in $\boldsymbol{\gamma}$, is

$$\boldsymbol{\gamma}[\text{SUBST}_p\ \boldsymbol{\alpha}] \equiv \boldsymbol{\gamma}[p \mapsto \boldsymbol{\alpha}] \qquad .$$

Under adjunction, an internal node of $\boldsymbol{\gamma}$ at p labeled $f \in \mathcal{F}$ is *split apart*, replaced by an auxiliary-form tree $\boldsymbol{\beta}$ rooted in f' when $|f| = |f'|$. The

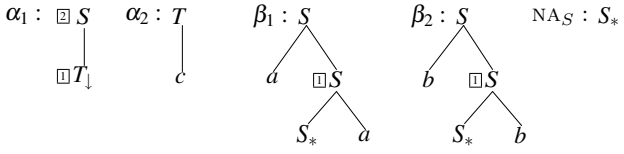


Figure 4:
Sample TAG for the copy language $\{wcw \mid w \in \{a,b\}^*\}$. The initial symbol is $S_\downarrow$.

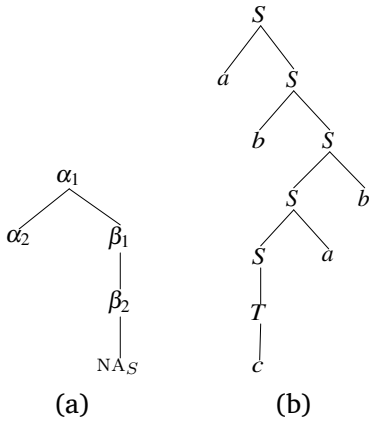


Figure 5:
Derivation and derived trees for the sample grammar of Figure 4: (a) derivation tree, (b) corresponding derived tree.

resulting tree, the adjunction of $\boldsymbol{\beta}$ at p in $\boldsymbol{\gamma}$, is

$$\boldsymbol{\gamma}[\textsc{adj}_p\ \boldsymbol{\beta}] \equiv \boldsymbol{\gamma}[p \mapsto \boldsymbol{\beta}[\text{foot}(\boldsymbol{\beta}) \mapsto \boldsymbol{\gamma}/p]] \qquad .$$

This definition (by requiring f to be in $\mathcal{F}$, not $\mathcal{F}_*$ or $\mathcal{F}_\downarrow$) is consistent with the standard convention, without loss of expressivity, that adjunction is disallowed at foot nodes and substitution nodes.

For uniformity, we will notate these operations with a single operator $\textsc{op}_p$ defined as follows:

$$\boldsymbol{\gamma}[\textsc{op}_p\ \boldsymbol{\gamma}'] \equiv \begin{cases} \boldsymbol{\gamma}[\textsc{subst}_p\ \boldsymbol{\gamma}'] & \text{if } \boldsymbol{\gamma}@p \in \mathcal{F}_\downarrow \\ \boldsymbol{\gamma}[\textsc{adj}_p\ \boldsymbol{\gamma}'] & \text{otherwise} \end{cases}$$

4.3 *Derivation trees and the derivation relation*

A derivation tree D records the operations over the elementary trees used to derive a given derived tree. Each node in the derivation tree specifies an elementary tree ${}_\square\gamma$, with the node's child subtrees D_i recording the derivations for trees that are adjoined or substituted into that tree at the corresponding operable nodes.

A DERIVATION for a grammar $G = \langle \mathcal{F}, P, S \rangle$ is a tree whose nodes are labeled with elementary trees, that is, a tree D in $\mathcal{T}(P)$. We here

interpret P itself as a ranked alphabet, where for each ${}_{\square}\gamma = \langle \gamma, \pi \rangle \in P$, we take its arity to be $arity({}_{\square}\gamma) \equiv |\pi|$. This requirement enforces the constraint that nodes in a derivation tree labeled with ${}_{\square}\gamma$ will have exactly the right number of children to specify the subtrees to be used at each of the operable sites in ${}_{\square}\gamma$. We add an additional constraint:

Labels match: For each node in D labeled with ${}_{\square}\gamma = \langle \gamma, \pi \rangle$, and for all i where $1 \leq i \leq arity({}_{\square}\gamma)$, the root node of the i-th child of ${}_{\square}\gamma$, labeled with ${}_{\square}\gamma_i$, must match the corresponding operable site in ${}_{\square}\gamma$, that is,

$$|\gamma @ \pi_i| = |\gamma_i @ \epsilon| \quad .$$

(The notation $\gamma @ \pi_i$ can be thought of as the node in γ labeled by diacritic $\boxed{i}$.)

A derivation is COMPLETE if it is rooted in an initial tree that is itself rooted in the initial symbol:

Initial symbol at root: The tree ${}_{\square}\alpha_r$ at the root of the derivation tree must be an initial tree labeled at its root by the initial symbol; that is, $|\alpha_r @ \epsilon| = |S|$.[5]

For example, the tree in Figure 5(a) is a well-formed complete derivation tree for the grammar in Figure 4. Note, for instance, that $|\alpha_1 @ \pi_2| = S = |\beta_1 @ \epsilon|$ as required by the label-matching constraint, and the root is an initial tree α_1 whose root is consistent with the initial symbol $S_{\downarrow}$.

A simple tree automaton can check these conditions, and therefore define the set of well-formed complete derivation trees. This automaton is constructed as follows. The states of the automaton are the set $\{q_N \mid N \in |\mathcal{F}|\}$, one for each unranked vocabulary symbol in the derived tree language. The start state is $q_{|S|}$. For each tree ${}_{\square}\gamma = \langle \gamma, \pi \rangle \in P$, of arity n and rooted with the symbol N, there is a transition of the form

$$q_{|N|}({}_{\square}\gamma(x_1, \ldots, x_n)) \doteq {}_{\square}\gamma(q_{|\gamma @ \pi_1|}(x_1), \ldots, q_{|\gamma @ \pi_n|}(x_n)) \quad . \quad (2)$$

The set of well-formed derivation trees is thus a regular tree set.

[5] The stripping of ranks and diacritics is necessary to allow, for instance, the initial symbol to match root nodes of differing arities.

For the grammar of Figure 4, the automaton defining well-formed derivation trees is given by

$$\begin{aligned} q_S(\alpha_1(x,y)) &\doteq \alpha_1(q_T(x), q_S(y)) \\ q_T(\alpha_2) &\doteq \alpha_2 \\ q_S(\beta_1(x)) &\doteq \beta_1(q_S(x)) \\ q_S(\beta_2(x)) &\doteq \beta_2(q_S(x)) \\ q_S(\mathrm{NA}_S) &\doteq \mathrm{NA}_S \end{aligned}$$

which recognizes the tree of Figure 5(a):

$$\begin{aligned} q_S(\alpha_1(\alpha_2, \beta_1(\beta_2(\mathrm{NA}_S)))) &\doteq \alpha_1(q_T(\alpha_2), q_S(\beta_1(\beta_2(\mathrm{NA}_S)))) \\ &\doteq \alpha_1(\alpha_2, \beta_1(q_S(\beta_2(\mathrm{NA}_S)))) \\ &\doteq \alpha_1(\alpha_2, \beta_1(\beta_2(q_S(\mathrm{NA}_S)))) \\ &\doteq \alpha_1(\alpha_2, \beta_1(\beta_2(\mathrm{NA}_S))) \end{aligned}$$

The DERIVATION RELATION $\mathcal{D}$, that is, the relation between derivation trees and the derived trees that they specify, can be simply defined via the hierarchical iterative operation of trees at operable sites. In particular, for a derivation tree with root labeled with the elementary tree ${}_{\square}\gamma = \langle \gamma, \pi \rangle$ of arity n, we define

$$\mathcal{D}({}_{\square}\gamma(t_1, \ldots, t_n)) \equiv \gamma[\mathrm{OP}_{\pi_1}\, \mathcal{D}(t_1), \mathrm{OP}_{\pi_2}\, \mathcal{D}(t_2), \ldots, \mathrm{OP}_{\pi_n}\, \mathcal{D}(t_n)]$$

where, following Schabes and Shieber (1994), the right-hand side specifies the *simultaneous* application of the specified operations. We define this in terms of the *sequential* application of operations as follows:

$$\begin{aligned} &\gamma[\mathrm{OP}_{p_1}\, \gamma_1, \mathrm{OP}_{p_2}\, \gamma_2, \ldots, \mathrm{OP}_{p_n}\, \gamma_n] \\ &\qquad \equiv \gamma[\mathrm{OP}_{p_1}\, \gamma_1][\mathrm{OP}_{update(p_2,\gamma_1,p_1)}\, \gamma_2, \ldots, \mathrm{OP}_{update(p_n,\gamma_1,p_1)}\, \gamma_n] \end{aligned} \quad (3)$$

The *update* function adjusts the paths at which later operations take place to compensate for an earlier adjunction. (Recall the notations $q \prec p$ for q a proper prefix of p and $p - q$ for the sequence obtained by removing the prefix q from p.)

$$update(p, \gamma, q) \equiv \begin{cases} p & \text{if } \gamma \text{ is an initial-form tree} \\ p & \text{if } \gamma \text{ is an auxiliary-form tree and } q \not\prec p \\ q \cdot \mathrm{foot}(\gamma) \cdot (p - q) & \\ & \text{if } \gamma \text{ is an auxiliary-form tree and } q \prec p \end{cases}$$

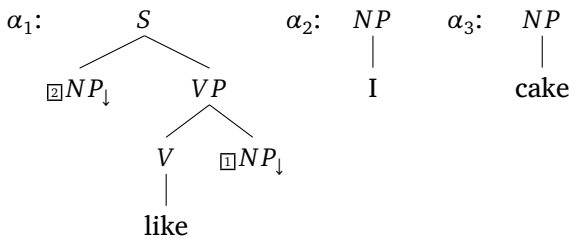


Figure 6: Grammar for a tiny English fragment.

Schabes and Shieber (1994) prove that adjunctions at distinct sites commute: if $p \neq q$ then

$$\gamma[\ldots, \text{ADJ}_p\ \gamma_1, \text{ADJ}_q\ \gamma_2, \ldots] = \gamma[\ldots, \text{ADJ}_q\ \gamma_2, \text{ADJ}_p\ \gamma_1, \ldots] \qquad (4)$$

that is, that the order of adjunctions is immaterial according to this definition. The proof applies equally well to substitution and mixtures of operations. This proves that the order of the permutation over operable sites is truly arbitrary; any order will yield the same result. (In Section 8, the introduction of multiple adjunction presents the potential for noncommutativity. We address the issue in that section.)

As the base case, this definition gives, as expected,

$$\mathcal{D}({}_{\square}\gamma) \equiv \gamma$$

for elementary trees of arity 0, that is, trees with no operable sites.

4.4 *Tree-substitution grammars*

Tree-substitution grammars are simply tree-adjoining grammars with no auxiliary trees, so that the elementary trees are only combined by substitution.

As a simple natural-language example, we consider the grammar with three elementary trees of Figure 6 and initial symbol S. The arities of the symbols should be clear from their usage and the associated permutations from the link diacritics.

As in Section 4.3, the derived tree for a derivation tree D is generated by performing all of the requisite substitutions. In this section, we provide a new definition of the derivation relation between a derivation tree and the derived tree it specifies as a simple homomorphism $h_{\mathcal{D}}$, and prove that this definition is equivalent to that of Section 4.3.

We define $h_{\mathcal{D}}$ in equational form. For each elementary tree ${}_{\square}\alpha \in P$, there is an equation of the form

$$h_{\mathcal{D}}({}_{\square}\alpha(x_1, \ldots, x_n)) \doteq \lfloor {}_{\square}\alpha \rfloor$$

where the right-hand-side transformation $\lfloor \cdot \rfloor$ is defined by

$$\begin{aligned} \lfloor A(t_1, \ldots, t_n) \rfloor &= A(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \\ \lfloor {}_{\boxed{k}}A_{\downarrow} \rfloor &= h_{\mathcal{D}}(x_k) \qquad . \end{aligned} \tag{5}$$

Essentially, this transformation replaces each operable site π_i by the homomorphic image of the corresponding variable x_i, that is,

$$\lfloor \alpha \rfloor = \alpha[\pi_1 \mapsto h_{\mathcal{D}}(x_1)] \ldots [\pi_n \mapsto h_{\mathcal{D}}(x_n)]$$

for a tree α with n substitution sites in its permutation π.

4.5 *An example derivation*

Returning to the example, the equations corresponding to the elementary trees of Figure 6 are

$$\begin{aligned} h_{\mathcal{D}}(\alpha_1(x_1, x_2)) &\doteq S(h_{\mathcal{D}}(x_2), VP(V(like), h_{\mathcal{D}}(x_1))) \\ h_{\mathcal{D}}(\alpha_2) &\doteq NP(I) \\ h_{\mathcal{D}}(\alpha_3) &\doteq NP(cake) \qquad . \end{aligned}$$

We define the derived tree corresponding to a derivation tree D as the application of this homomorphism to D, that is $h_{\mathcal{D}}(D)$. For the example above, the derived tree is that shown in Figure 2(a):

$$\begin{aligned} h_{\mathcal{D}}(\alpha_1(\alpha_3, \alpha_2)) &\doteq S(h_{\mathcal{D}}(\alpha_2), VP(V(like), h_{\mathcal{D}}(\alpha_3)) \\ &\doteq S(NP(I), VP(V(like), NP(cake))) \end{aligned}$$

By composing the automaton recognizing well-formed derivation trees with the homomorphism above, we can construct a single transducer doing the work of both. We do this explicitly for TAG in Section 7.1.

Note that, by construction, each variable occurs exactly once on the right-hand side of a given equation. Thus, this homomorphism $h_{\mathcal{D}}$ is linear and complete.

4.6 *Equivalence of $\mathcal{D}$ and $h_{\mathcal{D}}$*

We can show that this definition in terms of the linear complete homomorphism $h_{\mathcal{D}}$ is equivalent to the traditional definition $\mathcal{D}$:

$$\mathcal{D}(D) = h_{\mathcal{D}}(D) \tag{6}$$

The proof is by induction on the height of D. Since $h_{\mathcal{D}}$ is the identity function everywhere except at operable sites,

$$\mathcal{D}({}_{\square}\alpha) = \alpha = h_{\mathcal{D}}({}_{\square}\alpha) \quad .$$

This serves as the base case for the induction.

Now suppose, that Equation (6) holds for trees of height k, and consider tree ${}_{\square}\alpha(D_1, \ldots, D_n)$ of height $k+1$. Then

$$\begin{aligned}
\mathcal{D}({}_{\square}\alpha(D_1, \ldots, D_n)) &= \alpha[\textsc{subst}_{\pi_1}\, \mathcal{D}(D_1), \ldots, \textsc{subst}_{\pi_n}\, \mathcal{D}(D_n)] \\
&= \alpha[\textsc{subst}_{\pi_1}\, \mathcal{D}(D_1)] \ldots [\textsc{subst}_{\pi_n}\, \mathcal{D}(D_n)] \\
&= \alpha[\pi_1 \mapsto \mathcal{D}(D_1)] \ldots [\pi_n \mapsto \mathcal{D}(D_n)] \\
&= \alpha[\pi_1 \mapsto h_{\mathcal{D}}(D_1)] \ldots [\pi_n \mapsto h_{\mathcal{D}}(D_n)] \qquad \Leftarrow \\
&= \alpha[\pi_1 \mapsto h_{\mathcal{D}}(x_1)] \ldots [\pi_n \mapsto h_{\mathcal{D}}(x_n)][D_1, \ldots, D_n] \\
&= \lfloor \alpha \rfloor [D_1, \ldots, D_n] \\
&= h_{\mathcal{D}}({}_{\square}\alpha(D_1, \ldots, D_n)) \quad .
\end{aligned}$$

The marked step applies the induction hypothesis.

Later, in Section 7 we will provide a similar reformulation of the derivation relation for tree-adjoining grammars. To do so, however, requires additional power beyond simple tree homomorphisms, which is the subject of that section.

5 SYNCHRONOUS GRAMMARS

We perform synchronization of tree-adjoining and tree-substitution grammars as per the approach taken in earlier work (Shieber 1994). Synchronous grammars consist of pairs of elementary trees with a linking relation between operable sites in each tree. Simultaneous operations occur at linked nodes. In the case of synchronous tree-substitution grammars, the composition operation is substitution, so the linked nodes are substitution nodes.

We define a synchronous tree-adjoining grammar, then, as a quintuple $G = \langle \mathcal{F}_{in}, \mathcal{F}_{out}, P, S_{in}, S_{out} \rangle$, where

- $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ are the input and output ranked alphabets, respectively,
- $S_{in} \in \mathcal{F}_{in\downarrow}$ and $S_{out} \in \mathcal{F}_{out\downarrow}$ are the input and output initial symbols, and
- P is a set of elementary linked tree pairs, each of the form $\langle \gamma_{in}, \gamma_{out}, \frown \rangle$, where $\gamma_{in} \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{in\downarrow} \cup \mathcal{F}_{in*})$ and $\gamma_{out} \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{in\downarrow} \cup \mathcal{F}_{in*})$ are input and output trees and $\frown \subseteq \overline{\gamma_{in}} \times \overline{\gamma_{out}}$ is a bijection between operable sites from the two trees.

We define $G_{in} = \langle \mathcal{F}_{in}, P_{in}, S_{in} \rangle$ where $P_{in} = \{\langle \gamma, \pi_{in} \rangle \mid \langle \gamma, \gamma', \frown \rangle \in P\}$; this is the left projection of the synchronous grammar onto a simple TAG. The right projection G_{out} is defined similarly. Recall that the elementary trees in this grammar need a permutation on their operable sites. In order to guarantee that derivations for the synchronized grammars are isomorphic, the permutations for the operable sites for paired trees should be consistent. We therefore choose an arbitrary permutation $\langle p_{in,1} \frown p_{out,1}, \ldots, p_{in,n} \frown p_{out,n} \rangle$ over the linked pairs, and take the permutations π_{in} for γ_{in} and π_{out} for γ_{out} to be defined as $\pi_{in} = \langle p_{in,1}, \ldots, p_{in,n} \rangle$ and $\pi_{out} = \langle p_{out,1}, \ldots, p_{out,n} \rangle$. Since $\frown$ is a bijection, these projections are permutations as required.

A synchronous derivation was originally defined (Shieber 1994) as a pair $\langle D_{in}, D_{out} \rangle$ where[6]

1. D_{in} is a well-formed derivation tree for G_{in}, and D_{out} is a well-formed derivation tree for G_{out}, and
2. D_{in} and D_{out} are isomorphic.[7]

The derived tree pair for derivation $\langle D_{in}, D_{out} \rangle$ is then $\langle \mathcal{D}(D_{in}), \mathcal{D}(D_{out}) \rangle$.

[6] In our earlier definition (Shieber 1994), a third condition required that the isomorphic operations be sanctioned by links in tree pairs. This condition can be dropped here, as it follows from the previous definitions. In particular, since the permutations for paired trees are chosen to be consistent, it follows that the isomorphic children of isomorphic nodes are substituted at linked paths.

[7] By "isomorphism" here, we mean the normal sense of isomorphism of rooted trees where the elementary-tree-pairing relation in P serves as the bijection witnessing the isomorphism.

Presentations of synchronous tree-adjoining grammars typically weaken the requirement that the linking relation be a bijection; multiple links are allowed to impinge on a single node. One of two interpretations is possible in this case. We might require that if multiple links impinge upon a node, only one of the links be used. Under this interpretation, the multiple links at a node can be thought of as abbreviatory for a set of trees, each of which contains only one of the links. (The abbreviated form allows for exponentially fewer trees, however.) Thus, the formalism is equivalent to the one described in this section in terms of bijective link relations. Alternatively, we might allow true multiple adjunction of nontrivial trees, which requires an extended notion of derivation tree and derivation relation. This interpretation, proposed by Schabes and Shieber (1994), is arguably better motivated. We defer discussion of multiple adjunction to Section 8, where we address the issue in detail.

6 THE BIMORPHISM CHARACTERIZATION OF STSG

The central result we provide relating STSG to tree transducers is this: STSG is weakly equivalent to $B(LC, LC)$, that is, equivalent in the characterized string relations. To show this, we must demonstrate that any STSG is reducible to a bimorphism, and vice versa.

6.1 *Reducing STSG to* $B(LC, LC)$

Given an STSG $G = \langle \mathcal{F}_{in}, \mathcal{F}_{out}, P, S_{in}, S_{out} \rangle$, we need to construct a bimorphism characterizing the same tree relation. All the parts are in place to do this. We start by defining a language $\mathbb{D}$ of synchronous derivation trees, which recasts synchronous derivations as single derivation trees from which the left and right derivation trees can be projected via homomorphisms. Rather than taking a synchronous derivation to be a pair of isomorphic trees D_{in} and D_{out}, we take it to be the single tree D isomorphic to both, whose element at address p is the elementary tree pair in P that includes $D_{in}@p$ and $D_{out}@p$. The two synchronized derivations D_{in} and D_{out} can be separately recovered by projecting this new derivation tree on its first and second elements via homomorphisms: h_{in} that projects on the first component and h_{out} that projects on the second, respectively. These homomorphisms are trivially linear and complete (indeed, they are mere delabelings).

We define the set $\mathbb{D}$ of well-formed synchronous derivation trees to be the set of trees $D \in \mathcal{T}(P)$ such that $h_{in}(D)$ and $h_{out}(D)$ are both well-formed derivation trees as per Section 4.3. Since tree automata are closed under inverse homomorphism and intersection, the set $\mathbb{D}$ is a regular tree language.

The fact that for any tree $D \in \mathbb{D}$, $h_{in}(D)$ and $h_{out}(D)$ are well-formed derivation trees for their respective TSGs is trivial by construction. It is also trivial to show that any paired derivation has a corresponding synchronous derivation tree in $\mathbb{D}$.

For a given derivation tree $D \in \mathbb{D}$, the paired derived trees can be constructed as $h_{\mathcal{D}}(h_{in}(D))$ and $h_{\mathcal{D}}(h_{out}(D))$, respectively. Thus the mappings from the derivation tree to the derived trees are the compositions of two linear complete homomorphisms, hence linear complete homomorphisms themselves. We take the bimorphism characterizing the STSG tree relation to be $\langle \mathbb{D}, h_{\mathcal{D}} \circ h_{in}, h_{\mathcal{D}} \circ h_{out} \rangle$. Thus, the tree relation defined by the STSG is in $B(LC, LC)$.

6.2 *Reducing $B(LC, LC)$ to STSG*

The other direction is somewhat trickier to prove. Given a bimorphism $\langle L, h_{in}, h_{out} \rangle$ over input and output alphabets $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$, respectively, we construct a corresponding STSG $G = \langle \mathcal{F}'_{in}, \mathcal{F}'_{out}, P, S_{in}, S_{out} \rangle$. By "corresponding", we mean that the tree relation defined by the bimorphism is obtainable from the tree relation defined by the STSG via simple homomorphisms of the input and output that eliminate the nodes labeled in Q (as described below). The tree yields are unchanged by these homomorphisms; thus, the string relations defined by the bimorphism and the synchronous grammar are identical.

As the language L is a regular tree language, it is generable by a nondeterministic tree automaton $h_D = \langle Q, \mathcal{F}_d, \Delta, q_0 \rangle$. We use the states of this automaton in the input and output alphabets of the STSG. The input alphabet of the STSG is $\mathcal{F}'_{in} = \mathcal{F}_{in} \cup Q$, composed of the input symbols of the bimorphism, along with the states of the automaton (taken to be symbols of arity 1), and similarly for the output alphabet. The state symbols mark the places in the tree where substitutions occur, allowing control for appropriate substitutions. It is these state symbols that can be eliminated by a simple homomorphism.[8]

[8] In previous work (Shieber 2004), we used a construction that did not in-

The basic idea of the STSG construction is to construct an elementary tree pair corresponding to each compatible pair of transitions in the transducer $h_D \circ h_{in} = \langle Q_{in}, \mathcal{F}_d, \mathcal{F}_{in}, \Delta_{in}, q_{in,0} \rangle$ and $h_D \circ h_{out} = \langle Q_{out}, \mathcal{F}_d, \mathcal{F}_{out}, \Delta_{out}, q_{out,0} \rangle$. For each pair of transitions of the form

$$q_{in,i}(f(x_1, \ldots, x_n)) \doteq \tau_{in} \in \Delta_{in}$$

and

$$q_{out,j}(f(x_1, \ldots, x_n)) \doteq \tau_{out} \in \Delta_{out}$$

we construct a tree pair

$$\langle q_{in,i}(\lceil \tau_{in} \rceil), q_{out,j}(\lceil \tau_{out} \rceil) \rangle$$

where the following transformation is applied to the right-hand sides of the transitions to form the body of the synchronized trees:

$$\lceil f(t_1, \ldots, t_m) \rceil = f(\lceil t_1 \rceil, \ldots, \lceil t_m \rceil)$$
$$\lceil q_j(x_k) \rceil = \boxed{k} q_{j\downarrow}$$

Note that this transformation generates the tree along with a permutation of the operable sites (all substitution nodes) in the tree, and that there will be exactly n such sites in each element of the tree pair, since the transitions are linear and complete by hypothesis. Thus, the two permutations define an appropriate linking relation, which we take to be the synchronous grammar linking relation for the tree pair.

An example may clarify the construction. Take the language of the bimorphism to be defined by the following two-state automaton:

$$q(f(x, y)) \doteq f(q'(x), q'(y))$$
$$q(a) \doteq a$$
$$q'(g(x)) \doteq g(q(x))$$

troduce any extra tree structure in the STSG, so that the trees generated by the bimorphism relation could be recovered by a delabeling rather than a homomorphism deleting extra nodes. However, the proof of equivalence was considerably more subtle, and did not generalize as readily to the case of STAG. Nonetheless, it is useful to note that even more faithful STSG reconstructions of bimorphisms are possible.

Alternately, the definition of STSG (and similarly, STAG) can be modified to incorporate finite-state information explicitly at operable sites. By adding in this information, the bookkeeping done here can be folded into the states, allowing for a stricter strong-generative capacity equivalence. This elegant approach is pursued by Büchse *et al.* (2014).

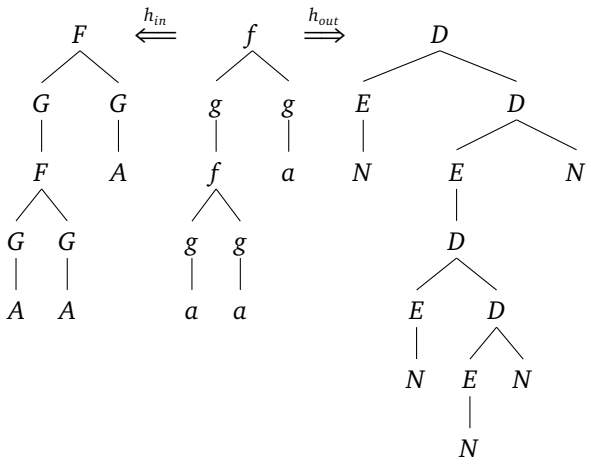


Figure 7: Example of bimorphism construction

This automaton uses the states to alternate g's with f's and a's level by level. For instance, it admits the middle tree in Figure 7. With input and output homomorphisms defined by

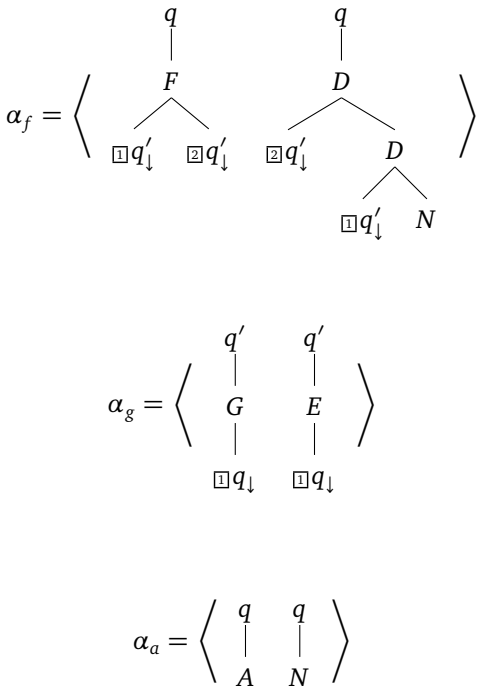
$$\hat{h}_{in}(f) \doteq F(x,y) \qquad \hat{h}_{out}(f) \doteq D(y, D(x,N))$$
$$\hat{h}_{in}(g) \doteq G(x) \qquad \hat{h}_{out}(g) \doteq E(x)$$
$$\hat{h}_{in}(a) \doteq A \qquad \hat{h}_{out}(a) \doteq N$$

the bimorphism so defined generates the tree relation instance exemplified in the figure.

The construction given above generates the elementary tree pairs in Figure 8 for this bimorphism. The reader can verify that the grammar generates a tree pair which corresponds to that shown in Figure 7 generated by the bimorphism after deletion of the state symbols.

By placing STSG in the class of bimorphisms, which have already been used to characterize tree transducers, we synthesize these two independently developed approaches to specifying tree relations. But the relation between a TAG derivation tree and its derived tree is not a mere homomorphism. The appropriate morphism generalizing linear complete homomorphisms to allow adjunction can be used to provide

Figure 8: Generated STSG for previous example of bimorphism construction (in Figure 7)



a bimorphism characterization of STAG as well, further unifying these strands of research. It is to this possibility that we now turn.

7 EMBEDDED TREE TRANSDUCERS

We have shown that the string relations defined by synchronous tree-substitution grammars were exactly the relations $B(LC, LC)$. Intuitively speaking, the tree language in such a bimorphism represents the set of derivation trees for the synchronous grammar, and each homomorphism represents the relation between the derivation tree and the derived tree for one of the projected tree-substitution grammars. The homomorphisms are linear and complete because the tree relation between a tree-substitution grammar derivation tree and its associated derived tree is exactly a linear complete tree homomorphism. To characterize the relations defined by synchronous tree-adjoining grammars, it similarly suffices to *find a simple homomorphism-like char-*

acterization of the tree relation between TAG derivation trees and derived trees. In Section 7.3 below, we show that linear complete embedded tree homomorphisms (which we introduce next) serve this purpose.

Embedded tree transducers are a generalization of tree transducers in which states are allowed to take a single additional argument in a restricted manner. They correspond to a restrictive subcase of macro tree transducers with one recursion variable. We use the term "embedded tree transducer" rather than the more cumbersome "monadic macro tree transducer" for brevity and by analogy with embedded pushdown automata (Schabes and Vijay-Shanker 1990), another automata-theoretic characterization of the tree-adjoining languages.

We modify the grammar of transducer equations to add an extra optional argument to each occurrence of a state q. To highlight the special nature of the extra argument, it is written in angle brackets before the input tree argument. We uniformly use the otherwise unused variable x_0 for this argument in the left-hand side, and add x_0 as a possible right-hand side itself. Finally, right-hand-side occurrences of states may be passed an arbitrary further right-hand-side tree in this argument. (The use of square brackets in the metanotation indicates optionality.)

$$
\begin{array}{rcl}
q \in Q & & \\
f^{(n)} \in \mathcal{F}^{(n)} & & \\
x \in \mathcal{X} & ::= & x_0 \mid x_1 \mid x_2 \mid \cdots \\
Eqn & ::= & q\langle[x_0]\rangle(f^{(n)}(x_1, \ldots, x_n)) \doteq \tau^{(n)} \\
\tau^{(n)} \in \mathcal{R}^{(n)} & ::= & f^{(m)}({\tau^{(n)}}_1, \ldots, {\tau^{(n)}}_m) \\
& \mid & x_0 \\
& \mid & q_j\langle[{\tau^{(n)}}_j]\rangle(x_i) \quad \text{where } 1 \leq i \leq n
\end{array}
\tag{7}
$$

Embedded transducers are strictly more expressive than traditional transducers, because the extra argument allows unbounded communication between positions unboundedly distant in depth in the output tree. For example, a simple embedded transducer can compute the reversal of a string, transducing $1(2(2(nil)))$ to $2(2(1(nil)))$, for instance. (This is not computable by a traditional tree transducer.) It is given by the following equations:

$$\begin{aligned} r\langle\rangle(nil) &\doteq nil \\ r\langle\rangle(1(x)) &\doteq s\langle 1(nil)\rangle(x) \\ r\langle\rangle(2(x)) &\doteq s\langle 2(nil)\rangle(x) \\ s\langle x_0\rangle(nil) &\doteq x_0 \\ s\langle x_0\rangle(1(x)) &\doteq s\langle 1(x_0)\rangle(x) \\ s\langle x_0\rangle(2(x)) &\doteq s\langle 2(x_0)\rangle(x) \end{aligned} \tag{8}$$

This is, of course, just the normal accumulating reverse functional program, expressed as an embedded transducer.[9] The additional power of embedded transducers is exactly what is needed to characterize the additional power that TAGs represent over CFGs in describing tree languages, as we will demonstrate in this section. In particular, we show that the relation between a TAG derivation tree and derived tree is characterized by a deterministic linear complete embedded tree transducer (DLCETT).

The first direct presentation of the connection between the tree-adjoining languages and macro tree transducers – the basis for the presentation here – was given in an earlier paper (Shieber 2006). However, the connection may be implicit in a series of previous results in the formal-language theory literature.[10] For instance, Fujiyoshi and Kasai (2000) show that linear, complete monadic context-free tree grammars generate exactly the tree-adjoining languages via a normal form for spine grammars. Separately, the relation between context-free tree grammars and macro tree transducers has been described, where the relationship between the monadic variants of each is implicit. Thus, taken together, an equivalence between the tree-adjoining

[9] A simpler set of equations achieves the same end.

$$\begin{aligned} r\langle\rangle(x) &\doteq s\langle nil\rangle(x) \\ s\langle x_0\rangle(nil) &\doteq x_0 \\ s\langle x_0\rangle(1(x)) &\doteq s\langle 1(x_0)\rangle(x) \\ s\langle x_0\rangle(2(x)) &\doteq s\langle 2(x_0)\rangle(x) \end{aligned} \tag{9}$$

Unfortunately, this set of equations doesn't satisfy the structure of an embedded tree transducer given in Equation (7). Surprisingly, however, the compilation from equations to TAG presented in Section 7.2 applies to this set of equations as well, generating a TAG whose derived trees also reverse its derivation trees.

[10] We are indebted to Uwe Mönnich for this observation.

languages and the image languages of monadic macro tree transducers might be pieced together.

In the present work, we define the relation between tree-adjoining languages and linear complete embedded tree transducers directly, simply, and transparently, by giving explicit constructions in both directions. First, we show that for any TAG we can construct a DLCETT that specifies the tree relation between the derivation trees for the TAG and the derived trees. Then, we show that for any DLCETT we can construct a TAG such that the tree relation between the derivation trees and derived trees is related through a simple homomorphism to the DLCETT tree relation. Finally, we use these results to show that STAG and the bimorphism class $B(ELC, ELC)$ are weakly equivalent, where ELC stands for the class of linear complete embedded homomorphisms.

7.1 *From TAG to transducer*

As the first part of the task of characterizing TAG in terms of DLCETT, we show that for any TAG grammar $G = \langle \mathcal{F}, P, S \rangle$, there is a DLCETT $\langle \{h_{\mathcal{D}}\}, P, \mathcal{F}, \Delta, h_{\mathcal{D}} \rangle$ (in fact, an embedded homomorphism), that transduces the derivation trees for the grammar to the corresponding derived trees. This transducer plays the same role for TAG as the definition of $h_{\mathcal{D}}$ in Section 4.3 did for TSG. We define the components of the transducer as follows: The single state, evocatively named $h_{\mathcal{D}}$, is the initial state. The input alphabet is the set of elementary trees P in the grammar, since the input trees are to be the derivation trees of the grammar. The arity of a tree (qua symbol in the input alphabet) is as described in Section 4.3. The output alphabet is that used to define the trees in the TAG grammar, $\mathcal{F}$.

We now turn to the construction of the equations, one for each elementary tree ${}_{\square}\gamma \in P$. Suppose ${}_{\square}\gamma$ has a permutation $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ on its operable sites. (We use this ordering by means of the diacritic representation below.) If γ is an auxiliary tree, construct the equation

$$h_{\mathcal{D}}\langle x_0 \rangle({}_{\square}\gamma(x_1, \ldots, x_n)) \doteq \lfloor \gamma \rfloor$$

and if γ is an initial tree, construct the equation

$$h_{\mathcal{D}}\langle\rangle({}_{\square}\gamma(x_1, \ldots, x_n)) \doteq \lfloor \gamma \rfloor$$

where the right-hand-side transformation $\lfloor\cdot\rfloor$ is defined by[11]

$$\begin{aligned}
\lfloor f(t_1,\ldots,t_n)\rfloor &= f(\lfloor t_1\rfloor,\ldots,\lfloor t_n\rfloor)\\
\lfloor \boxed{k} f(t_1,\ldots,t_n)\rfloor &= h_{\mathcal{D}}\langle\lfloor f(t_1,\ldots,t_n)\rfloor\rangle(x_k)\\
\lfloor f_*\rfloor &= x_0\\
\lfloor \boxed{k} f_{\downarrow}\rfloor &= h_{\mathcal{D}}\langle\rangle(x_k) \qquad .
\end{aligned} \tag{10}$$

Note that the equations so generated are linear and complete, because each variable x_i is generated once as the tree α is traversed, namely at position π_i in the traversal (marked with $\boxed{i}$), and the variable x_0 is generated at the foot node only. Thus, the generated embedded tree transducer is linear and complete. Because only one equation is generated per tree, the transducer is trivially deterministic. Because there is only one state, it is a kind of embedded homomorphism.

As noted for TSG in Section 4.3, by composing the automaton recognizing well-formed derivation trees from Section 4.3 with the embedded homomorphism above generating the derived tree, we can construct a single DLCETT doing the work of both. Where the construction of Section 4.3 would generate a transition of the form in Equation 2, repeated here as

$$q_{|N|}({}_{\square}\gamma(x_1,\ldots,x_n)) \doteq {}_{\square}\gamma(q_{|\gamma@\pi_1|}(x_1),\ldots,q_{|\gamma@\pi_n|}(x_n))$$

we compose this transition with the corresponding transition from the previous section

$$h_{\mathcal{D}}\langle x_0\rangle({}_{\square}\gamma(x_1,\ldots,x_n)) \doteq \lfloor\gamma\rfloor$$

[11] It may seem like trickery to use the diacritics in this way, as they are not really components of the tree being traversed, but merely manifestations of an extrinsic ordering. But their use is benign. The same transformation can be defined, a bit more cumbersomely, keeping the permutation π separate, by tracking the permutation and the current address p in a revised transformation $\lfloor\cdot\rfloor_{\pi,p}$ defined as follows:

$$\begin{aligned}
\lfloor f(t_1,\ldots,t_n)\rfloor_{\pi,p} &= f(\lfloor t_1\rfloor_{\pi,p\cdot 1},\ldots,\lfloor t_n\rfloor_{\pi,p\cdot n})\\
\lfloor f(t_1,\ldots,t_n)\rfloor_{\pi,p} &= h_{\mathcal{D}}\langle\lfloor f(t_1,\ldots,t_n)\rfloor_{\pi,p}\rangle(x_{\pi^{-1}(p)})\\
\lfloor f_*\rfloor_{\pi,p} &= x_0\\
\lfloor f_{\downarrow}\rfloor_{\pi,p} &= h_{\mathcal{D}}\langle\rangle(x_{\pi^{-1}(p)})
\end{aligned}$$

We then use $\lfloor\alpha\rfloor_{\pi,\epsilon}$ for the transformation of the tree α.

or

$$h_{\mathcal{D}}\langle\rangle({}_{\square}\gamma(x_1,\ldots,x_n)) \doteq \lfloor\gamma\rfloor$$

for auxiliary and initial trees respectively. The composition construction generates a transducer with states in the cross-product of the states of the input transducers. In this case, since the latter transducer has a single state, we simply reuse the state set of the former, generating

$$q_{|N|}\langle x_0\rangle({}_{\square}\gamma(x_1,\ldots,x_n)) \doteq \lfloor\gamma\rfloor$$

or

$$q_{|N|}\langle\rangle({}_{\square}\gamma(x_1,\ldots,x_n)) \doteq \lfloor\gamma\rfloor$$

where

$$\begin{aligned}
\lfloor f(t_1,\ldots,t_n)\rfloor &= f(\lfloor t_1\rfloor,\ldots,\lfloor t_n\rfloor)\\
\lfloor \boxed{k} f(t_1,\ldots,t_n)\rfloor &= q_{|f|}\langle\lfloor f(t_1,\ldots,t_n)\rfloor\rangle(x_k)\\
\lfloor f_*\rfloor &= x_0\\
\lfloor \boxed{k} f_{\downarrow}\rfloor &= q_{|f_{\downarrow}|}\langle\rangle(x_k) \qquad .
\end{aligned} \tag{11}$$

7.1.1 An example derivation

By way of example, we consider the tree-adjoining grammar given by the following trees:

$$\begin{aligned}
\alpha: &\quad \boxed{1}A(e)\\
\beta_A: &\quad A(\boxed{1}B(a),\boxed{2}C(\boxed{3}D(A_*)))\\
\beta_B: &\quad \boxed{1}B(b,B_*)\\
\mathrm{NA}_B: &\quad B_*\\
\mathrm{NA}_C: &\quad C_*\\
\mathrm{NA}_D: &\quad D_*
\end{aligned}$$

Starting with the auxiliary tree $\beta_A = A(\boxed{1}B(a),\boxed{2}C(\boxed{3}D(A_*)))$, the adjunction sites, corresponding to the nodes labeled B, C, and D at addresses 1, 2, and 21, have been arbitrarily given a preorder permu-

tation. We therefore construct the equation as follows:

$$\begin{aligned}
h_{\mathcal{D}}\langle x_0\rangle(\beta_A(x_1,x_2,x_3)) &\doteq \lfloor A(\boxed{1}B(a),\boxed{2}C(\boxed{3}D(A_*)))\rfloor \\
&= A(\lfloor \boxed{1}B(a)\rfloor, \lfloor \boxed{2}C(\boxed{3}D(A_*))\rfloor) \\
&= A(h_{\mathcal{D}}\langle \lfloor B(a)\rfloor\rangle(x_1), \lfloor \boxed{2}C(\boxed{3}D(A_*))\rfloor) \\
&= A(h_{\mathcal{D}}\langle B(\lfloor a\rfloor)\rangle(x_1), \lfloor \boxed{2}C(\boxed{3}D(A_*))\rfloor) \\
&= \cdots \\
&= A(h_{\mathcal{D}}\langle B(a)\rangle(x_1), h_{\mathcal{D}}\langle C(h_{\mathcal{D}}\langle D(x_0)\rangle(x_3))\rangle(x_2))
\end{aligned}$$

Similar derivations for the remaining trees yield the (deterministic linear complete) embedded tree transducer defined by the following set of equations:

$$\begin{aligned}
h_{\mathcal{D}}\langle\rangle(\alpha(x_1)) &\doteq h_{\mathcal{D}}\langle A(e)\rangle(x_1) \\
h_{\mathcal{D}}\langle x_0\rangle(\beta_A(x_1,x_2,x_3)) &\doteq A(h_{\mathcal{D}}\langle B(a)\rangle(x_1), h_{\mathcal{D}}\langle C(h_{\mathcal{D}}\langle D(x_0)\rangle(x_3))\rangle(x_2)) \\
h_{\mathcal{D}}\langle x_0\rangle(\beta_B(x_1)) &\doteq h_{\mathcal{D}}\langle B(b,x_0)\rangle(x_1) \\
h_{\mathcal{D}}\langle x_0\rangle(\mathrm{NA}_B()) &\doteq x_0 \\
h_{\mathcal{D}}\langle x_0\rangle(\mathrm{NA}_C()) &\doteq x_0 \\
h_{\mathcal{D}}\langle x_0\rangle(\mathrm{NA}_D()) &\doteq x_0
\end{aligned}$$

We can use this transducer to compute the derived tree for the derivation tree

$$\alpha(\beta_A(\beta_B(\mathrm{NA}_B),\mathrm{NA}_C,\mathrm{NA}_D))$$

as follows:

$$\begin{aligned}
&h_{\mathcal{D}}\langle\rangle(\alpha(\beta_A(\beta_B(\mathrm{NA}_B),\mathrm{NA}_C,\mathrm{NA}_D))) \\
&\quad\doteq h_{\mathcal{D}}\langle A(e)\rangle(\beta_A(\beta_B(\mathrm{NA}_B),\mathrm{NA}_C,\mathrm{NA}_D)) \\
&\quad\doteq A(h_{\mathcal{D}}\langle B(a)\rangle(\beta_B(\mathrm{NA}_B)), h_{\mathcal{D}}\langle C(h_{\mathcal{D}}\langle D(A(e))\rangle(\mathrm{NA}_D))\rangle(\mathrm{NA}_C)) \\
&\quad\doteq A(h_{\mathcal{D}}\langle B(b,B(a))\rangle(\mathrm{NA}_B), C(h_{\mathcal{D}}\langle D(A(e))\rangle(\mathrm{NA}_D))) \\
&\quad\doteq A(B(b,B(a)), C(D(A(e))))
\end{aligned}$$

7.1.2 Equivalence of $\mathcal{D}$ and $h_{\mathcal{D}}$

We can now show for TAG derivations, as we did for TSG derivations in Section 4.3, that the embedded homomorphism $h_{\mathcal{D}}$ constructed in this way computes the derivation relation $\mathcal{D}$.

In order to simplify the argument, we take advantage of the commutativity of operations (Equation 4), and assume without loss of generality that each permutation associated with the operable sites of an elementary tree is consistent with a postorder traversal of the nodes in the tree. We can then simplify Equation 3 to

$$\gamma[\mathrm{OP}_{p_1}\,\gamma_1, \mathrm{OP}_{p_2}\,\gamma_2, \ldots, \mathrm{OP}_{p_n}\,\gamma_n] \equiv \gamma[\mathrm{OP}_{p_1}\,\gamma_1][\mathrm{OP}_{p_2}\,\gamma_2, \ldots, \mathrm{OP}_{p_n}\,\gamma_n]$$

since in a postorder traversal, $p_i \not\prec p_{i+k}$.

It will also prove to be useful to have a single notation for the effect of both substitution and adjunction. Recall the definitions of substitution and adjunction:

$$\begin{aligned}\gamma[\mathrm{SUBST}_p\,\boldsymbol{\alpha}] &\equiv \gamma[p \mapsto \boldsymbol{\alpha}]\\ \gamma[\mathrm{ADJ}_p\,\boldsymbol{\beta}] &\equiv \gamma[p \mapsto \boldsymbol{\beta}[\mathrm{foot}(\boldsymbol{\beta}) \mapsto \gamma/p]]\end{aligned}$$

Under the convention that mapping a (nonexistent) "foot" of an initial tree leaves the tree unchanged, that is,

$$\boldsymbol{\alpha}[\mathrm{foot}(\boldsymbol{\alpha}) \mapsto \gamma] \equiv \boldsymbol{\alpha}$$

the two operations collapse notationally, so that we can write

$$\gamma[\mathrm{OP}_p\,\gamma'] \equiv \gamma[p \mapsto \gamma'[\mathrm{foot}(\gamma') \mapsto \gamma/p]]$$

for both substitution and adjunction.

We prove that $\mathcal{D}(D) = h_{\mathcal{D}}\langle\rangle(D)$ for derivations D rooted in an initial tree, and $\mathcal{D}(D)[\mathrm{foot}(\mathcal{D}(D)) \mapsto x] = h_{\mathcal{D}}\langle x\rangle(D)$ for derivations rooted in an auxiliary tree. The proof is again by induction on the height of the derivation D.

For the base case, the derivation consists of a single tree with no operable sites. If it is an initial tree α, then $\mathcal{D}(\alpha) = \alpha = h_{\mathcal{D}}\langle\rangle(\alpha)$ straightforwardly from the definition of $h_{\mathcal{D}}$, using only the first equation in Equation (10). Similarly, the base case for auxiliary trees,

$$\mathcal{D}(\beta)[\mathrm{foot}(\beta) \mapsto x] = \beta[\mathrm{foot}(\beta) \mapsto x] = h_{\mathcal{D}}\langle x\rangle(\beta)$$

requires only the first and third equations in (10).

For the recursive case,

$$
\begin{aligned}
&h_{\mathcal{D}}\langle\rangle(\alpha(D_1,\ldots,D_n))\\
&\quad = \lfloor\alpha\rfloor[x_1 \mapsto D_1,\ldots,x_n \mapsto D_n]\\
&\quad = \alpha[\pi_1 \mapsto h_{\mathcal{D}}\langle\alpha/\pi_1\rangle(D_1)]\cdots[\pi_n \mapsto h_{\mathcal{D}}\langle\alpha/\pi_n\rangle(D_n)]\\
&\quad = \alpha[\pi_1 \mapsto \mathcal{D}(D_1)[\text{foot}(\mathcal{D}(D_1)) \mapsto \alpha/\pi_1]]\\
&\qquad\quad \cdots[\pi_1 \mapsto \mathcal{D}(D_n)[\text{foot}(\mathcal{D}(D_n)) \mapsto \alpha/\pi_n]] \qquad\qquad \Leftarrow\\
&\quad = \alpha[\text{OP}_{\pi_1}\ \mathcal{D}(D_1)]\cdots[\text{OP}_{\pi_n}\ \mathcal{D}(D_n)]\\
&\quad = \alpha[\text{OP}_{\pi_1}\ \mathcal{D}(D_1),\cdots,\text{OP}_{\pi_n}\ \mathcal{D}(D_n)]\\
&\quad = \mathcal{D}(\alpha(D_1,\ldots,D_n))
\end{aligned}
$$

with the marked step appealing to the induction hypothesis.

Similarly, for derivations rooted in an auxiliary tree,

$$
\begin{aligned}
&h_{\mathcal{D}}\langle x\rangle(\beta(D_1,\ldots,D_n))\\
&\quad = \lfloor\beta\rfloor[x_0 \mapsto x, x_1 \mapsto D_1,\ldots,x_n \mapsto D_n]\\
&\quad = \beta[\text{foot}(\beta) \mapsto x][\pi_1 \mapsto h_{\mathcal{D}}\langle\beta/\pi_1\rangle(D_1)]\\
&\qquad\quad \cdots[\pi_n \mapsto h_{\mathcal{D}}\langle\beta/\pi_n\rangle(D_n)]\\
&\quad = \beta[\text{foot}(\beta) \mapsto x][\pi_1 \mapsto \mathcal{D}(D_1)[\text{foot}(\mathcal{D}(D_1)) \mapsto \beta/\pi_1]]\\
&\qquad\quad \cdots[\pi_1 \mapsto \mathcal{D}(D_n)[\text{foot}(\mathcal{D}(D_n)) \mapsto \beta/\pi_n]]\\
&\quad = \beta[\text{foot}(\beta) \mapsto x][\text{OP}_{\pi_1}\ \mathcal{D}(D_1)]\cdots[\text{OP}_{\pi_n}\ \mathcal{D}(D_n)]\\
&\quad = \beta[\text{foot}(\beta) \mapsto x][\text{OP}_{\pi_1}\ \mathcal{D}(D_1),\ldots,\text{OP}_{\pi_n}\ \mathcal{D}(D_n)]\\
&\quad = \beta[\text{OP}_{\pi_1}\ \mathcal{D}(D_1),\cdots,\text{OP}_{\pi_n}\ \mathcal{D}(D_n)][\text{foot}(\mathcal{D}(\beta(D_1,\ldots,D_n))) \mapsto x]\\
&\quad = \mathcal{D}(\beta(D_1,\ldots,D_n))[\text{foot}(\mathcal{D}(\beta(D_1,\ldots,D_n))) \mapsto x] \qquad .
\end{aligned}
$$

7.2 *From transducer to TAG*

Having shown how to construct a DLCETT that captures the relation between derivation trees and derived trees of a TAG, we turn now to showing how to construct a TAG that mimics in its derivation/derived tree relation a DLCETT. Given a linear complete embedded tree transducer $\langle Q, \mathcal{F}, \mathcal{G}, \Delta, q_0\rangle$, we construct a corresponding TAG $\langle \mathcal{G} \cup \dot{Q}, P, \dot{q}_0\rangle$ where the alphabet consists of the output alphabet $\mathcal{G}$ of the transducer together with the disjoint set of unary symbols $\dot{Q} = \{\dot{q}_1,\ldots,\dot{q}_{|Q|}\}$ corresponding to the states of the input transducer. The initial symbol of

the grammar is the symbol $\dot{q}_0$ corresponding to the initial state q_0 of the transducer.

The elementary trees of the grammar are constructed as follows. For each rule of the form

$$q\langle[x_0]\rangle(f^{(m)}(x_1,\ldots,x_m)) \doteq \tau$$

we build a tree named $\langle q, f, \tau\rangle$. Where this tree appears is determined solely by the state q, so we take the root node of the tree to be the corresponding symbol $\dot{q}$. Any foot node in the tree will also need to be marked with the same label, so we pass this information down as the tree is built inductively. The tree is therefore of the form $\dot{q}(\lceil\tau\rceil_q)$ where the right-hand-side transformation $\lceil\cdot\rceil_q$ constructs the remainder of the tree by the inductive walk of τ, with the subscript noting that the root is labeled by state q.

$$\begin{aligned}
\lceil f^{(m)}(t_1,\ldots,t_m)\rceil_q &= f(\lceil t_1\rceil_q,\ldots,\lceil t_m\rceil_q)\\
\lceil q_j\langle\tau\rangle(x_k)\rceil_q &= \boxed{k}\dot{q}_j(\lceil\tau\rceil_q)\\
\lceil q_j\langle\rangle(x_k)\rceil_q &= \boxed{k}\dot{q}_{j\downarrow}\\
\lceil x_0\rceil_q &= \dot{q}_*
\end{aligned}$$

Note that at x_0, a foot node is generated of the proper label. (Because the equation is linear, only one foot node is generated, and it is labeled appropriately by construction.) Where recursive processing of the input tree occurs ($q_j\langle\tau\rangle(x_k)$), we generate a tree that admits adjunctions at $\dot{q}_j$. The role of the diacritic $\boxed{k}$ is merely to specify the permutation of operable sites for interpreting derivation trees; it says that the k-th child in a derivation tree rooted in the current elementary tree is taken to specify adjunctions at this node.

The trees generated by this TAG correspond to the outputs of the corresponding tree transducer. Because of the more severe constraints on TAG, in particular that all combinatorial limitations on putting subtrees together must be manifest in the labels in the trees themselves, the outputs actually contain more structure than the corresponding transducer output. In particular, the state-labeled nodes are merely for bookkeeping. A simple homomorphism removing these nodes gives

the desired transducer output:[12]

$$\begin{aligned} rem(\dot{q}(x)) &\doteq rem(x) && \text{for } \dot{q} \in \dot{Q} \\ rem(f^{(n)}(x_1, \ldots, x_n)) &\doteq f^{(n)}(rem(x_1), \ldots, rem(x_n)) && \text{for } f^{(n)} \in \mathcal{G}^{(n)} \end{aligned}$$

An example may clarify the construction. Recall the reversal embedded transducer in (8) above. The construction above generates a TAG containing the following trees. We have given them indicative names rather than the cumbersome ones of the form $\langle q_i, f, \tau \rangle$.

$$\begin{aligned} \alpha_{nil}: &\quad \dot{r}(nil) \\ \alpha_1: &\quad \dot{r}(\boxed{1}\dot{s}(1(nil))) \\ \alpha_2: &\quad \dot{r}(\boxed{1}\dot{s}(2(nil))) \\ \beta_{nil}: &\quad \dot{s}(\dot{s}_*) \\ \beta_1: &\quad \dot{s}(\boxed{1}\dot{s}(1(\dot{s}_*))) \\ \beta_2: &\quad \dot{s}(\boxed{1}\dot{s}(2(\dot{s}_*))) \end{aligned}$$

It is simple to verify that the derivation tree

$$\alpha_1(\beta_2(\beta_2(\beta_{nil})))$$

derives the tree

$$\dot{r}(\dot{s}^4(2(\dot{s}(2(\dot{s}(1(nil))))))) \quad .$$

Simple homomorphisms that extract the input function symbols on the input and drop the bookkeeping states on the output (that is, the homomorphism *rem* provided above) reduce these trees to $1(2(2(nil)))$ and $2(2(1(nil)))$ respectively, just as for the corresponding tree transducer.

7.2.1 Equivalence of DLCETT and TAG

We demonstrate that the compilation from DLCETT to TAG generates a grammar with the same language as that of the DLCETT by appeal to the previous result of Section 7.1.2. Consider a DLCETT $T = \langle Q, \mathcal{F}, \mathcal{G}, \Delta, q_0 \rangle$ converted by the compilation above to a grammar $G = \langle \mathcal{G} \cup \dot{Q}, P, \dot{q}_0 \rangle$. That grammar may itself be compiled to a

[12] As noted in Footnote 8, a formalization of a modified form of TAG that directly incorporates state information at operable sites (Büchse *et al.* 2014) eliminates this need for bookkeeping through extra nodes in the tree structure, making the equivalence even stronger.

DLCETT using the compilation of Section 7.1.2, previously shown to be language-preserving. We show that this round-trip conversion preserves the language that is the range of the DLCETT by showing that each equation in the original grammar "round-trip" compiles to an equation that differs only in the tree structure. In particular, a rule of the form $q\langle x_0\rangle(f(x_1,\ldots,x_m)) = \tau$ compiles to the equation $q\langle x_0\rangle(f(x_1,\ldots,x_m)) = \tau'$ where $\tau = rem(\tau')$. We will write $\tau' \approx \tau$ when $\tau = rem(\tau')$.

For each rule in T of the form $q\langle x_0\rangle(f(x_1,\ldots,x_m)) = \tau$, we generate a tree $\langle q, f, \tau\rangle$ in the grammar G of the form $\dot{q}(\lceil\tau\rceil_q)$. This tree, in turn, is compiled as in Section 7.1 to an equation in the output transducer T':

$$\begin{aligned} q\langle x_0\rangle(\langle q, f, \tau\rangle(x_1,\ldots,x_m)) &= \lfloor\dot{q}(\lceil\tau\rceil_q)\rfloor \\ &= \dot{q}(\lfloor\lceil\tau\rceil_q\rfloor) \\ &\approx \lfloor\lceil\tau\rceil_q\rfloor \end{aligned}$$

(Here and in the following, we write q for $q_{|\dot{q}|}$ in the $\lfloor\cdot\rfloor$ construction, taking advantage of the bijection between the $\dot{Q}$ symbols and the corresponding states of the generated transducer.) Note that this is exactly of the required form, so long as $\lfloor\lceil\tau\rceil_q\rfloor \approx \tau$, which we now prove by induction on the structure of τ.

- If $\tau = x_0$, $\lfloor\lceil x_0\rceil_q\rfloor = \lfloor\dot{q}_*\rfloor = x_0$.
- If $\tau = q_j\langle\rangle(x_k)$, $\lfloor\lceil q_j\langle\rangle(x_k)\rceil_q\rfloor = \lfloor\boxed{k}\dot{q}_{j\downarrow}\rfloor = q_j\langle\rangle(x_k)$.
- If $\tau = q_j\langle\tau_0\rangle(x_k)$,

  $$\begin{aligned} \lfloor\lceil q_j\langle\tau_0\rangle(x_k)\rceil_q\rfloor &= \lfloor\boxed{k}\dot{q}_j(\lceil\tau_0\rceil_q)\rfloor \\ &= q_j\langle\lfloor\dot{q}_j(\lceil\tau_0\rceil_q)\rfloor\rangle(x_k) \\ &= q_j\langle\dot{q}_j(\lfloor\lceil\tau_0\rceil_q\rfloor)\rangle(x_k) \\ &\approx q_j\langle\tau_0\rangle(x_k). \end{aligned}$$

  The last step follows from the induction hypothesis and the fact that rem removes the symbol $\dot{q}_j$.
- If $\tau = f^{(m)}(t_1,\ldots,t_m)$,

  $$\begin{aligned} \lfloor\lceil f^{(m)}(t_1,\ldots,t_m)\rceil_q\rfloor &= \lfloor f^{(m)}(\lceil t_1\rceil_q,\ldots,\lceil t_m\rceil_q)\rfloor \\ &= f^{(m)}(\lfloor\lceil t_1\rceil_q\rfloor,\ldots,\lfloor\lceil t_m\rceil_q\rfloor) \\ &\approx f^{(m)}(t_1,\ldots,t_m). \end{aligned}$$

Again, the last step applies the induction hypothesis.

Writing $L(T)$ for the range string language of the transducer T, we have that $L(G) = L(T')$ and $L(T) = L(T')$. We conclude that $L(T) = L(G)$. In fact, by the above, the tree languages are identical up to the homomorphism *rem*. Most importantly, then, the weak generative capacity of TAGs and the range of DLCETTs are identical.

7.3 *The bimorphism characterization of STAG*

The major advantage of characterizing TAG derivation in terms of tree transducers (via the compilation (10)) is the integration of synchronous TAGs into the bimorphism framework, which follows directly.

In order to model a synchronous grammar formalism as a bimorphism, the well-formed derivations of the synchronous formalism must be characterizable as a regular tree language and the relation between such derivation trees and each of the paired derived trees as a homomorphism of some sort. As shown in Section 6, for synchronous tree-substitution grammars, derivation trees are regular tree languages, and the map from derivation to each of the paired derived trees is a linear complete tree homomorphism. Thus, synchronous tree-substitution grammars fall in the class of bimorphisms $B(LC, LC)$. The other direction holds as well; all bimorphisms in $B(LC, LC)$ define string relations expressible by an STSG.

A similar result follows for STAG. Crucially relying on the result above that the derivation relation is a DLCETT, we can use the same method directly to characterize the synchronous TAG string relations as just $B(ELC, ELC)$. We have thus integrated synchronous TAG with the other transducer and synchronous grammar formalisms falling under the bimorphism umbrella.

8 MULTIPLE ADJUNCTION

The discussion so far has assumed that derivations allow at most one operation to occur at any given node in an elementary tree (in fact, exactly one). This constraint inhered in the original formulations of TAG derivation (Vijay-Shanker 1987), and had the effect of removing systematic spurious ambiguities without reducing the range of defin-

able languages. Schabes and Shieber (1994) point out the desirability of allowing multiple adjunctions at a single node, and provide various arguments for this generalization, most notably as needed for many applications of synchronous TAG, which is precisely the case that we are concerned with in this paper. It therefore behooves us to examine the effect of multiple adjunction on the analysis.

There are various ways in which multiple adjunction can be inserted. Most simply, one could specify that the set of operable nodes of a tree allows for a given node in the set a fixed number of times. (This could be graphically depicted by allowing more than one diacritic at a given node, with each diacritic to be used exactly once.) In theory, this would allow multiple nontrivial adjunctions to occur at a single node, inducing ambiguity as to the resulting derived tree, but we can eliminate this possibility by requiring that nontrivial (that is, non-NA) trees be adjoined at at most one site at a given node. We start by handling this kind of simple generalization of TAG derivation in Sections 8.1–8.2.

More generally, Schabes and Shieber (1994) call for allowing an arbitrary number of adjunctions at a given node. In particular, they call for distinguishing predicative and modifier auxiliary trees, and allowing any number of modifier trees and at most one predicative tree to adjoin at a given node. The derived tree is ambiguous as to the relative orderings of the modifier trees, but the predicative tree is required to fall above the modifier trees. We address this major generalization of TAG derivation in Section 8.3.

8.1 *Simple multiple adjunction*

We start with a simple generalization of TAG derivation in which operable nodes may be used a fixed number of times. Since the set of operable nodes may now include duplicates, adjunction nodes may occur more than once in the permutation π. To guarantee that at most one of these can be nontrivially adjoined, we need to revise the definition of derivation tree, that is, fix the tree automaton from Section 4.3 defining well-formed derivation trees, and to prove that the derivation relation $\mathcal{D}$ is still well-defined.

We present an alternative automaton defining the regular tree language of well-formed derivation trees now allowing the limited form of multiple adjunction. We double the number of states from

the previous construction. The states of the automaton are the set $\{q_{N\triangle} \mid N \in \mathcal{F}\} \cup \{q_{N\bullet} \mid N \in \mathcal{F}\}$, two for each unranked vocabulary symbol in the derived tree language. The $\triangle$ diacritic indicates a nontrivial tree rooted in the given symbol; the $\bullet$ diacritic requires a nonadjunction NA tree rooted in that symbol. The start state is $q_{|S|\triangle}$.

For each nontrivial tree (that is, not an NA tree) ${}_{\square}\gamma = \langle \gamma, \pi \rangle$, of arity n and rooted with the symbol N, we construct all possible transitions of the form

$$q_{|N|\triangle}(\gamma(x_1, \ldots, x_n)) \doteq \gamma(q_1(x_1), \ldots, q_n(x_n))$$

where each q_i is either $q_{|\gamma@\pi_i|\bullet}$ or $q_{|\gamma@\pi_i|\triangle}$, subject to the constraint that for each node η in α, the sequence $\langle q_i \mid \pi_i = \eta \rangle$ contains at most one $\triangle$. Because there are many such ways of setting the q_i to satisfy this constraint, there are many (though still a finite number of) transitions for each γ.

In addition, for NA trees, there is a transition

$$q_{|N|\bullet}(\gamma) \doteq \gamma \qquad .$$

The set of well-formed derivation trees is thus still a regular tree set.

The only remaining issue is to verify that the limited form of multiple adjunction that we allow still yields a well-defined derived tree. In general, multiple adjunctions at the same site do not commute. However, the only cases of multiple adjunctions that we allow involve all but one of the auxiliary trees being vestigial nonadjunction trees. Such cases do commute. It suffices to show that $\boldsymbol{\gamma}[\text{ADJ}_p\ \boldsymbol{\beta}, \text{ADJ}_p\ \text{NA}] = \boldsymbol{\gamma}[\text{ADJ}_p\ \text{NA}, \text{ADJ}_p\ \boldsymbol{\beta}]$; we derive this as follows:

$$\begin{aligned}
\boldsymbol{\gamma}[\text{ADJ}_p\ \boldsymbol{\beta}, \text{ADJ}_p\ \text{NA}] &= \boldsymbol{\gamma}[\text{ADJ}_p\ \boldsymbol{\beta}][\text{ADJ}_{update(p,\boldsymbol{\beta},p)}\ \text{NA}] \\
&= \boldsymbol{\gamma}[\text{ADJ}_p\ \boldsymbol{\beta}][\text{ADJ}_p\ \text{NA}] \\
&= \boldsymbol{\gamma}[\text{ADJ}_p\ \boldsymbol{\beta}] \\
&= \boldsymbol{\gamma}[\text{ADJ}_p\ \text{NA}][\text{ADJ}_p\ \boldsymbol{\beta}] \\
&= \boldsymbol{\gamma}[\text{ADJ}_p\ \text{NA}][\text{ADJ}_{update(p,\boldsymbol{\beta},p)}\ \boldsymbol{\beta}] \\
&= \boldsymbol{\gamma}[\text{ADJ}_p\ \text{NA}, \text{ADJ}_p\ \boldsymbol{\beta}]
\end{aligned}$$

8.2 *Fixed multiple adjunction*

What if we allow more than one of the multiple (fixed) occurrences of a node to be operated on by a nontrivial auxiliary tree? At that point,

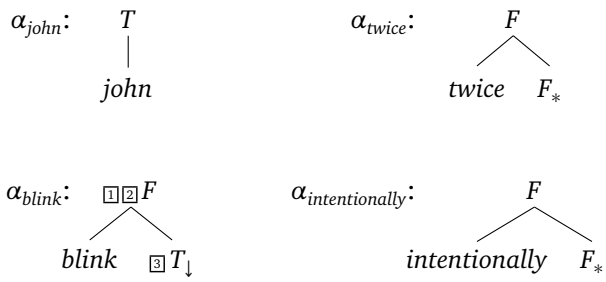


Figure 9:
A fragment with multiple adjunction.

the definition of simultaneous operations no longer commutes, and which auxiliary tree is used at which position becomes important.

The definition of the derivation tree language given in Section 4.3 allows such derivations to be specified merely by relaxing the constraint that a node appears only once in the set of operable sites. If we move to a multiset of operable sites, with π a permutation over that multiset, the remaining definitions generalize properly.

We present (Figure 9) a fragment based on the semantic half of a synchronous TAG presented previously (Shieber 1994, Figure 1) to exemplify simultaneous adjunction. This grammar uses simultaneous adjunction at the root of the α_{blink} tree. That tree has three operable sites, two of which are the root node. We will take the permutation of operable sites for the tree to be $\langle \boxed{1}, \boxed{2}, \boxed{3} \rangle$.

We can examine what the compilation of Section 7.1 provides as the interpretation for this grammar. Applying it to the output trees in the grammar generates a DLCETT. We start with the problematic multiple adjunction tree α_{blink}.

$$\begin{aligned} q_F\langle\rangle(\alpha_{blink}(x_1,x_2,x_3)) &\doteq \lfloor \alpha_{blink} \rfloor \\ &= \lfloor \boxed{1}\boxed{2}F(blink, \boxed{3}T) \rfloor \\ &= q_F\langle \lfloor \boxed{2}F(blink, \boxed{3}T) \rfloor \rangle(x_1) \\ &= q_F\langle q_F\langle \lfloor F(blink, \boxed{3}T) \rfloor \rangle(x_1) \rangle(x_2) \\ &= q_F\langle q_F\langle F(blink, q_T\langle\rangle(x_3)) \rangle(x_1) \rangle(x_2) \end{aligned}$$

(Here, the second line uses the obvious generalization of the second equation of (10) to sets of diacritics, that is,

$$\lfloor \boxed{k} \cdots f(t_1,\ldots,t_n) \rfloor = q_{|f|}\langle \lfloor \cdots f(t_1,\ldots,t_n) \rfloor \rangle(x_k) \quad ,$$

the ellipses standing in for arbitrary further diacritics.)

The second and third steps are notable here, in that the choice of which of the two operable sites to use first was arbitrary. That is, one could just as well have chosen to process diacritic [2] before [1], in which case the generated rule would have been

$$q_F\langle\rangle(\alpha_{blink}(x_1, x_2, x_3)) \doteq q_F\langle q_F\langle F(blink, q_T\langle\rangle(x_3))\rangle(x_2)\rangle(x_1) \quad .$$

This is, of course, just the consequence of the fact that multiple adjunctions at the same node do not commute. To manifest the ambiguity, we can just generate both transitions (and in general, all such transitions) in the transducer defining the derivation relation. The transducer naturally becomes nondeterministic. Alternatively, a particular order might be stipulated, regaining determinism, but giving up analyses that take advantage of the ambiguity.

Completing the compilation, we generate transitions for the other trees:

$$\begin{aligned} q_T\langle\rangle(\alpha_{john}) &\doteq T(john) \\ q_F\langle x_0\rangle(\beta_{twice}) &\doteq F(twice, x_0) \\ q_F\langle x_0\rangle(\beta_{intentionally}) &\doteq F(intentionally, x_0) \end{aligned}$$

The derivation tree $\alpha_{blink}(\beta_{intentionally}, \beta_{twice}, \alpha_{john})$ then derives trees as follows:

$$\begin{aligned} &q_F\langle\rangle(\alpha_{blink}(\beta_{intentionally}, \beta_{twice}, \alpha_{john})) \\ &\quad \doteq q_F\langle q_F\langle F(blink, q_T\langle\rangle(\alpha_{john}))\rangle(\beta_{intentionally})\rangle(\beta_{twice}) \\ &\quad \doteq q_F\langle q_F\langle F(blink, T(john))\rangle(\beta_{intentionally})\rangle(\beta_{twice}) \\ &\quad \doteq q_F\langle F(intentionally, F(blink, T(john)))\rangle(\beta_{twice}) \\ &\quad \doteq F(twice, F(intentionally, F(blink, T(john)))) \end{aligned}$$

corresponding to the meaning $twice(intentionally(blink(john)))$. Alternatively, use of the other nondeterministic alternative transition yields

$$\begin{aligned} &q_F\langle\rangle(\alpha_{blink}(\beta_{intentionally}, \beta_{twice}, \alpha_{john})) \\ &\quad \doteq q_F\langle q_F\langle F(blink, q_T\langle\rangle(\alpha_{john}))\rangle(\beta_{twice})\rangle(\beta_{intentionally}) \\ &\quad \doteq q_F\langle q_F\langle F(blink, T(john))\rangle(\beta_{twice})\rangle(\beta_{intentionally}) \\ &\quad \doteq q_F\langle F(twice, F(blink, T(john)))\rangle(\beta_{intentionally}) \\ &\quad \doteq F(intentionally, F(twice, F(blink, T(john)))) \end{aligned}$$

giving the alternative reading for the sentence.

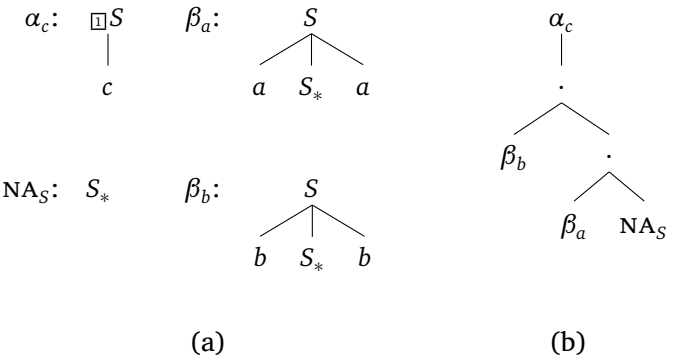


Figure 10:
A grammar (a) for $\{wcw^R \mid w \in \{a, b\}^*\}$ using general multiple adjunction, and (b) a derivation of the string $abba$.

8.3 *General multiple adjunction*

Finally, fully general multiple adjunction as described by Schabes and Shieber (1994) allows for one and the same operable site to be used an arbitrary number of times. To enable this interpretation of TAG derivations, major changes need to be made to the definitions of derivation tree and derivation relation.

Consider the sample grammar of Figure 10 where the two auxiliary trees β_a and β_b are modifier trees (in the terminology of Schabes and Shieber (1994)) and thus allowed to multiply adjoin at the two operable nodes in the initial tree. This grammar should generate the language $\{wcw^R \mid w \in \{a, b\}^*\}$.

Derivation trees must allow an arbitrary number of operations to occur at a given site. To represent this in a ranked tree, we can encode the sequence of trees adjoined at a given location with a recursive structure. In particular, we use a binary symbol $\cdot$ (which we write infix) to build a list of trees to be adjoined at the site, using a nonadjunction tree to mark the end of the list. Essentially, derivation trees now contain lists of auxiliary trees to operate at a site rather than a single tree, with the nonadjoining trees serving as the *nil* elements of the list and the binary $\cdot$ serving as the binary *constructor*. For example, a derivation for the grammar of Figure 10(a) can be represented by the tree in Figure 10(b).

The derivation tree language with lists instead of individual trees is still regular. In fact, the full specification of multiple adjunction given by Schabes and Shieber (1994) specifies that at a given operable site an arbitrary number of modifier trees but at most one predicative

tree may be adjoined. Further, the predicative tree is to appear highest in the derived tree above the adjoined modifiers. This constraint can be specified by defining the derivation tree language appropriately, allowing at most one predicative tree, and placing it at the end of the list of nontrivial trees adjoining at a site. It is a simple exercise to show that the derivation tree language so restricted still falls within the regular tree languages.

Finally, we must provide a definition of the derivation relation for this generalized form of multiple adjunction. In particular, we need transitions for the new form of constructor node, which specifies the combination of two adjunctions at a single site. We handle this by stacking the rest of the adjunctions above the first. We add to the definition of the derivation transducer of Section 7.1 transitions of the following form for each symbol N that is the root of some auxiliary tree:

$$q_N\langle x_0\rangle(x_1 \cdot x_2) \doteq q_N\langle q_N\langle x_0\rangle(x_1)\rangle(x_2)$$

Note that the new transition is still linear and complete.

For the grammar of Figure 10(a) we would thus have the following transitions defining the derivation relation:

$$\begin{aligned}
q_S\langle\rangle(\alpha(x)) &\doteq q_S\langle S(c)\rangle(x)\\
q_S\langle x_0\rangle(\mathrm{NA}_S) &\doteq x_0\\
q_S\langle x_0\rangle(\beta_a) &\doteq S(a, x_0, a)\\
q_S\langle x_0\rangle(\beta_b) &\doteq S(b, x_0, b)\\
q_S\langle x_0\rangle(x_1 \cdot x_2) &\doteq q_S\langle q_S\langle x_0\rangle(x_1)\rangle(x_2)
\end{aligned}$$

Using this derivation relation, the derived tree for the derivation tree of Figure 10(b) can be calculated as

$$\begin{aligned}
q_S\langle\rangle(\alpha_c(\beta_b \cdot \beta_a \cdot \mathrm{NA}_S)) &\doteq q_S\langle S(c)\rangle(\beta_b \cdot \beta_a \cdot \mathrm{NA}_S)\\
&\doteq q_S\langle q_S\langle S(c)\rangle(\beta_b)\rangle(\beta_a \cdot \mathrm{NA}_S)\\
&\doteq q_S\langle S(b, S(c), b)\rangle(\beta_a \cdot \mathrm{NA}_S)\\
&\doteq q_S\langle q_S\langle S(b, S(c), b)\rangle(\beta_a)\rangle(\mathrm{NA}_S)\\
&\doteq q_S\langle S(a, S(b, S(c), b), a)\rangle(\mathrm{NA}_S)\\
&\doteq S(a, S(b, S(c), b), a)
\end{aligned}$$

corresponding to the string $abcba$ as expected.

9 CONCLUSION

Synchronous grammars and tree transducers – two approaches to the specification of language relations useful for a variety of formal and computational linguistics modeling of natural languages – are unified by means of the elegant construct of the bimorphism. This convergence synthesizes the approaches and allows a direct comparison among these and other potential systems for describing language relations through other bimorphisms. The examination of additional bimorphism classes may open up further possibilities for useful modeling tools for natural language.

ACKNOWLEDGEMENTS

This paper has been gestating for a long time. I thank the participants in my course on Transducers at the 2003 European Summer School on Logic, Language, and Information in Vienna, Austria, where some of these ideas were presented, and Mark Dras, Mark Johnson, Uwe Mönnich, Rani Nelken, Rebecca Nesson, James Rogers, and Ken Shan for helpful discussions on the topic of this paper and related topics. The extensive comments of the JLM reviewers were invaluable in improving the paper. This work was supported in part by grant IIS-0329089 from the National Science Foundation.

REFERENCES

Alfred V. Aho and Jeffrey D. Ullman (1969), Syntax Directed Translations and the Pushdown Assembler, *Journal of Computer and System Sciences*, 3(1):37–56, doi:10.1016/S0022-0000(69)80006-1.

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas (2000), Learning Dependency Translation Models as Collections of Finite State Head Transducers, *Computational Linguistics*, 26(1):45–60, doi:10.1162/089120100561629.

André Arnold and Max Dauchet (1982), Morphismes et bimorphismes d'arbres [Morphisms and bimorphisms of trees], *Theoretical Computer Science*, 20(1):33–93, doi:10.1016/0304-3975(82)90098-6.

Matthias Büchse, Andreas Maletti, and Heiko Vogler (2012), Unidirectional Derivation Semantics for Synchronous Tree-Adjoining Grammars, in *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pp. 368–379, Springer, doi:10.1007/978-3-642-31653-1_33.

Matthias Büchse, Heiko Vogler, and Mark-Jan Nederhof (2014), Tree Parsing for Tree-Adjoining Machine Translation, *Journal of Logic and Computation*, 24(2):351–373, doi:10.1093/logcom/exs050.

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi (2008), Tree Automata Techniques and Applications, `http://tata.gforge.inria.fr/`, release of November 18, 2008.

Steve DeNeefe and Kevin Knight (2009), Synchronous Tree Adjoining Machine Translation, in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pp. 727–736, Association for Computational Linguistics, Singapore, `http://aclweb.org/anthology/D09-1076`.

Akio Fujiyoshi and Takumi Kasai (2000), Spinal-Formed Context-Free Tree Grammars, *Theory of Computing Systems*, 33:59–83, doi:10.1007/s002249910004.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu (2004), What's In a Translation Rule, in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pp. 273–280, Association for Computational Linguistics, Boston, Massachusetts, `http://aclweb.org/anthology/N04-1035`.

Jonathan Graehl and Kevin Knight (2004), Training Tree Transducers, in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pp. 105–112, Association for Computational Linguistics, Boston, Massachusetts, `http://aclweb.org/anthology/N04-1014`.

Chung-Hye Han and Nancy Hedberg (2008), Syntax and Semantics of It-Clefts: A Tree Adjoining Grammar Analysis, *Journal of Semantics*, 25:345–380, doi:10.1093/jos/ffn007.

Aravind Joshi and Yves Schabes (1997), Tree-Adjoining Grammars, in G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pp. 69–124, Springer, Berlin.

Alexander Koller and Marco Kuhlmann (2011), A Generalized View on Parsing and Translation, in *Proceedings of the 12th International Conference on Parsing Technologies*, IWPT '11, pp. 2–13, Association for Computational Linguistics, Stroudsburg, PA, USA, ISBN 978-1-932432-04-6, `http://dl.acm.org/citation.cfm?id=2206329.2206331`.

Philip M. Lewis II and Richard E. Stearns (1968), Syntax-Directed Transduction, *Journal of the Association for Computing Machinery*, 15(3):465–488, ISSN 0004-5411, doi:10.1145/321466.321477.

Andreas MALETTI (2008), Compositions of Extended Top-down Tree Transducers, *Information and Computation*, 206(9-10):1187–1196, doi:10.1016/j.ic.2008.03.019.

Andreas MALETTI, Jonathan GRAEHL, Mark HOPKINS, and Kevin KNIGHT (2009), The power of extended top-down tree transducers, *SIAM Journal on Computing*, 39:410–430, doi:10.1137/070699160.

I. Dan MELAMED (2003), Multitext Grammars and Synchronous Parsers, in *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 79–86, Association for Computational Linguistics, Edmonton, Canada, doi:10.3115/1073445.1073466.

I. Dan MELAMED (2004), Statistical Machine Translation by Parsing, in *Proceedings of the 42nd Annual Conference of the Association for Computational Linguistics*, pp. 653–660, Association for Computational Linguistics, Barcelona, Spain, doi:10.3115/1218955.1219038.

Mark-Jan NEDERHOF and Heiko VOGLER (2012), Synchronous Context-Free Tree Grammars, in *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, pp. 55–63, Paris, France.

Rebecca NESSON and Stuart M. SHIEBER (2006), Simpler TAG Semantics Through Synchronization, in *Proceedings of the 11th Conference on Formal Grammar*, pp. 129–142, Center for the Study of Language and Information, Malaga, Spain, `http://nrs.harvard.edu/urn-3:HUL.InstRepos:2252595`.

Rebecca NESSON, Stuart M. SHIEBER, and Alexander RUSH (2006), Induction of Probabilistic Synchronous Tree-Insertion Grammars for Machine Translation, in *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006)*, pp. 128–137, Association for Machine Translation in the Americas, Cambridge, Massachusetts, `http://nrs.harvard.edu/urn-3:HUL.InstRepos:2261232`.

William C. ROUNDS (1970), Mappings and Grammars on Trees, *Mathematical Systems Theory*, 4(3):257–287, doi:10.1007/BF01695769.

Yves SCHABES and Stuart M. SHIEBER (1994), An Alternative Conception of Tree-Adjoining Derivation, *Computational Linguistics*, 20(1):91–124, `http://aclweb.org/anthology/J94-1004`.

Yves SCHABES and K. VIJAY-SHANKER (1990), Deterministic Left to Right Parsing of Tree Adjoining Languages, in *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pp. 276–283, Association for Computational Linguistics, Pittsburgh, Pennsylvania, doi:10.3115/981823.981858.

Stuart M. Shieber (1994), Restricting the Weak-Generative Capacity of Synchronous Tree-Adjoining Grammars, *Computational Intelligence*, 10(4):371–385, doi:10.1111/j.1467-8640.1994.tb00003.x.

Stuart M. Shieber (2004), Synchronous Grammars as Tree Transducers, in *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pp. 88–95, Vancouver, Canada, http://nrs.harvard.edu/urn-3:HUL.InstRepos:2019322.

Stuart M. Shieber (2006), Unifying Synchronous Tree-Adjoining Grammars and Tree Transducers via Bimorphisms, in *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, pp. 377–384, European Chapter of the Association for Computational Linguistics, Trento, Italy, http://nrs.harvard.edu/urn-3:HUL.InstRepos:2252609.

Stuart M. Shieber and Yves Schabes (1990), Synchronous Tree-Adjoining Grammars, in *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pp. 253–258, International Committee on Computational Linguistics, Helsinki, Finland, doi:10.3115/991146.991191.

K. Vijay-Shanker (1987), *A Study of Tree Adjoining Grammars*, Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, http://repository.upenn.edu/dissertations/AAI8804974/.

Dekai Wu (1996), A Polynomial-Time Algorithm for Statistical Machine Translation, in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pp. 152–158, Association for Computational Linguistics, Santa Cruz, California, doi:10.3115/981863.981884.

Dekai Wu (1997), Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora, *Computational Linguistics*, 23(3):377–404, http://aclweb.org/anthology/J97-3002.

Elif Yamangil and Stuart M. Shieber (2010), Bayesian Synchronous Tree-Substitution Grammar Induction and Its Application to Sentence Compression, in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 937–947, Association for Computational Linguistics, Uppsala, Sweden, http://nrs.harvard.edu/urn-3:HUL.InstRepos:4733833.